Self Organized Multi Agent Swarms (SOMAS) for Network Security Control

THESIS

Eric M. Holloway, First Lieutenant, USAF

AFIT/GCS/ENG/09-02

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GCS/ENG/09-02

Self Organized Multi Agent Swarms (SOMAS) for Network Security Control

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the

Degree of Master of Science

Eric M. Holloway, B.S.C.S

First Lieutenant, USAF

March 2009

AFIT/GE/ENG/09-02

SELF ORGANIZED MULTI AGENT SWARMS (SOMAS) FOR NETWORK SECURITY CONTROL

Eric M. Holloway, B.S.C.S
First Lieutenant, USAF

Approved:

[illegible] 13 Mar 09
Dr. Gary Lamont, PhD (Chairman) Date

[illegible] 16 MAR 09
Dr. Gilbert Peterson, PhD (Member) Date

[illegible] 13 MAR 09
J. Todd McDonald, Lt Col, USAF (Member) Date

AFIT/GCS/ENG/09-02

Abstract

With the increasing number of stealthy computer network threats computer security has become a serious concern of commercial, industrial, and military organizations. Detection approaches for each of these general threats are of course very important security techniques in cyberspace warfare environments. Reactive activities are also required in order to respond to intrusions and anomalies, take protective actions, and possibly find the source of the attack and use corrective measures in these environments. A dynamic, distributed multi-agent model is suggested as a method of addressing these problems and generating the desired behaviors.

Contemporary security multi-agent structures are based on static, hierarchical designs which lack scalability and flexibility due to centralization of control and communication. Thus, self organized, entangled hierarchy multi-agent swarms are evolved in this study to decentralize control and communication.

In order to define the complexity of these network security problem domains, the formalized problem domains are mapped to partially observable Markov decision process (POMDP) mathematical models. Agent swarm and scenario specific behaviors are formalized within DEC-POMDP, I-POMDP, and a new F(*-POMDP) model in order to provide mathematical insight into the tractability of their complexity. This complexity implies that stochastic, global search for behavior solutions is the only computationally feasible solution technique.

The formalized agent swarm model is integrated with a new architectural solution for dynamically generating the desired behaviors computationally using an offline and online search technique. The resulting design exhibits a self organized multi-agent swarm (SOMAS) with entangled hierarchical control and communication through the use of multi-objective evolutionary algorithm optimization techniques. The development of this new and innovative architecture employs an a priori generation of agent structures (rules and parameters) based upon desired swarm behaviors. Then, online the swarm self-adapts to its environment using an online search.

SOMAS effectiveness is compared across multiple network scenarios using techniques of targeting intrusions, engaging intrusions, assessing intrusions and anomalies, and reacting to intrusions. Significant results across various behaviors and scenarios are described using statistical testing techniques for an appropriate variety of metrics. Also, significant results are shown in multi-objective visualization through Pareto-front graphs and analysis along with swarm simulation animation. Suggested future investigationsn as discussed should provide a more effective and efficient SOMAS for addressing network security problems.

Acknowledgements

Above all, I want to thank Jesus Christ for inspiring this thesis. Without Him, this life means nothing and I hope that my work, though it falls extremely short of being worthy of Him, gives the reader a glimpse of Who He is and what He means. I admit it is mysterious how this can be, but remember: He is God, the Logos, the principle of all things who is also a conscious, living Person, in whom we live, move, and have our being [John 1:1 & Acts 17:28]. What does it mean for such a Person to have come to earth to offer us salvation for our sins and rescue us from Hell? It is a historical fact, but also much, much more. Infinitely more than we could ever imagine.

The support of my family during this difficult time has been invaluable, and I deeply appreciate it. It is hard to express what it means to *always* have someone there for me. Professor Lamont has provided his expert guidance throughout, interesting discussions, and has given me free reign to explore and develop my ideas. Dr. Gilbert Peterson and Col McDonald both gave helpful feedback on the final document and presentation. I also thank my small group for their support and encouragement. The Brainstormers discussion group provided valuable insight and critique that shaped the crucial SOMAS concepts.

Eric M. Holloway

Table of Contents

Page

Page

Page

List of Figures

List of Tables

Self Organized Multi Agent Swarms (SOMAS) for Network Security Control

I. The Need for Decentralized Network Security Control

1.1 Introduction

Today, anyone can fire up a hacking script and take out their anger on their neighbor. The internet is in a state of war. A secret hacking battle between the US and China has been reported [113]. Russian computers have disrupted Estonia with denial of service attacks [136]. Al Queda extensively uses the internet for gaining support and organizing its operations [129]. The bitter, and ancient Sunni/Shiite feud has spilled over into modernity where websites have been defaced with threats and webservers have been shut down [13] [14]. Sweden and Turkey have fought each other online over the caricature of Muhammed in Swedish newspapers [12]. The proliferation of attacks has lead some to question whether cyber vigilante-ism might be a good idea [26]. The Air Force has made repeated, recent statements that it is no longer content with merely playing the defensive game, that reactivity is not an option [63] [20] [124]. It is not just nations that are involved. With the increasing number of stealthy computer network threats, computer security has become a serious concern of commercial, industrial, and military organizations from financial activities and power system operations to internet information communication, Cybercraft, and aircraft reconnaissance activities.

To stop the network security and stability threat important security tools are needed such as outside intrusion detection systems, insider covert network detection, resource protection, system anomaly detection techniques, attack reaction, and information surveillance. Many such systems have been proposed using standard hierarchical management structures with identification of features employing classical pattern recognition algorithms. Also used for network security are bio-inspired approaches from artificial immune system constructs [68] to particle swarm approaches [42] with support vector machines [84], as well as incorporating multi-objective aspects [39]. Various commercial security packages embed many of the associated algorithms. Nevertheless, such network security systems are generally quite slow and limited when operated automatically due to brittle hierarchical communication and control, and lack of autonomy.

Network security threats run the gambit from internal to external attacks. The most serious threats come from insider attacks, denial of service attacks, and information exploitation. These attacks aim at exploiting and corrupting information and capabilities on the networks, and the deactivation of network defense. The online battlefield requires a security solution that flexibly adapts in real-time to the constant flux of cyber war, while still providing overall control with humans possibly in the loop. All of this must be achieved without centralized control.

Currently, the Air Force is developing a defensive infrastructure by implementing secure middleware across all network hosts [65]. This defensive architecture is secure but rigid and hierarchical. Research shows that mobile, multi agent systems can provide the needed decentralized flexibility for such security systems, see Section 2.3.4. The solution is found in nature: extensive, vast self organized systems exist that semi-autonomously accomplish incredible complex tasks of resource management, defense, and offense without an overarching leader or internal blueprint.

Self organization is exhibited by many physical systems, ranging from the optical [50], to [70], to biological [66]. Additionally, self organization is a real phenomena according to mathematics and computer science [101]. This research explores how self organization allows the swarm behavior to adapt and respond to its environment in a decentralized manner without hierarchical control, using entangled hierarchies.

1.2 Goals and Objectives

The research goal is to investigate network security real-time performance using a swarm of autonomous self organized agents that evolve a non-hierarchical entangled Cyberspace security management structure.

In particular, this research defines a self-organized multi-agent swarm (SOMAS) system with a limited number of desired behaviors. A mobile, multi agent system that uses self organization to create entangled hierarchies is as or more robust, adaptive, and overall effective and efficient for accomplishing network based behaviors than systems that do not use self organization and entangled hierarchies. Therefore, the general objectives to show measurable completion of the goal are:

1. Create a robust simulation framework for evolving self organizing multi-agent systems (SOMAS)
   (a) Approach/foundation
      - Environment: formally defined computational network
      - Agent domain: formally defined mobile multi-agent swarm
      - Formal modeling: augmented POMDP models
      - Search: multi-objective evolutionary algorithms (MOEA)
      - Evaluation: simulation and metrics

   (b) Design
      - Two level search
      - Evolving self organized swarms
      - Intrinsic entangled hierarchy communication and control
      - Pareto front optimization
2. Evaluate feasibility of self organization for effective accomplishment of desired behaviors
   (a) Approach/foundation

        - Pareto plots
        - Behavior visualization

    (b) Design

        - Pareto plot objective correlation analysis
        - Visual identification of self organization

3. Evaluate feasibility of entangled hierarchies for effective accomplishment of desired behaviors

    (a) Approach/foundation

        - Pareto compliant metrics
        - Non-parametric hypothesis tests

    (b) Design

        - Pairwise swarm behavior hypothesis testing

1.3 SOMAS Architecture Design Approach

The off-line approach makes use of simulations to evaluate generated agents for a finite set of network security behaviors using random network configurations. This is represented by a decentralized partially observable Markov decision process (DEC-POMDP) [22] model. This model is chosen because the global network state and reward function can be observed for evaluating the global fitness of the swarm during off-line generation.

The on-line approach uses local data from the network the swarm is on, and thus an interactive partially observable Markov decision process (I-POMDP) [40] represents the on-line approach. The I-POMDP model is chosen because it presupposes individual agents are only capable of local observations of other agents in the swarm, and can only observe a local reward function. Furthermore, due to the inability of the POMDP models to fully represent a dynamic problem domain, the models are augmented. The augmentation replaces the state transition function with a model transition function. Thus, the models are capable of representing and adapting to dynamic systems. However, the POMDP models are not used for producing the swarm's rules. Both the off-line and on-line approaches make use of evolutionary operators to achieve optimum or optimal agent rules and parameters. The off-line production uses an explicit evolutionary algorithm and the on-line production's evolutionary algorithm is implicit in the interaction of the SOMAS and the environment. The swarm behaviors are formalized as reward functions for computational processing. The formalization consists of mathematical equations and well formed first order predicate logic formul æ.

1.4 Measuring Success

The attainment of these objectives can be achieved through a computational and stochastic measurement evaluation along with a visualization simulation of the results of the "optimization." Therefore, additional, derivative objectives are:

1. Develop new appropriate metrics for evaluating SOMAS behaviors
2. Use metrics to measure achievement of behaviors
3. Visualize swarm behaviors to qualify behavior characteristics

The behaviors are evaluated in two ways: statistical tests and visualization. The statistical tests give statistical quantifications of performance, while visualization gives a more qualitative analysis. Visual analysis consists of specifying visually identifiable characteristics, and then determining whether they exist within the swarm visualization.

1.5 Risks

Since the solution technique is stochastic, the major risk is due to the approximate nature of solutions. Approximation implies that there is imperfection, and consequently the SOMA swarm may not react appropriately to a given scenario. A second risk is due to the degree of self organization the swarms exhibit. If they do not remain self organized, then the swarm can either become trivial or chaotic, and therefore useless in both cases. Finally, the complexity and seriousness of the threats in Cyberspace may mean that the SOMA approach requires additional architectural modification.

1.6 Assumptions and Constraints

- The model developed for the agents assumes limited domain complexity. The agents are assumed to operate within a container model, which is a valid assumption if they are to be deployed in the CyberCraft structure.
- The agent structure consists of sensors, rules, and actuators. Additionally, the agent contains a memory that can carry different state information relevant for the agent, such as a fitness value.
- The OSI stack, as well as bandwidth and latency issues, are not addressed for network communication. Communication is simplified to direct delivery between nodes, i.e. an agent is merely moved from one node's data-structure and placed in another's data-structure. This means the agents are only being represented on each node, not on any intermediary edges unlike an actual network.
- The agents cannot fully trust each other due to the stochastic environment and the limitation of partial observations of the environment and each other. This constrains the POMDP model representing the agent swarm to a decentralized model, such as DEC-POMDP or I-POMDP, since complete swarm state information is not collectively available.

1.7 Innovation

The general innovative elements of this research are

- Development and implementation of a framework for the multi-objective evolution and testing of agent swarms.

- Quantified self organization
- Entangled hierarchies
- Most of the scenario behaviors, except for intrusion detection, are innovative

1.8 Thesis Organization

Section II describes the background research for the SOMAS problem domain, as well as symbolically describing the problem and algorithm domain. Section III breaks down the problem domain into its representation and complexity, and describes the design choices for creating the solution. Section IV explains the low level engineering of the design choices to create a solution. Section IV specifies the hypotheses to be tested and how the solution will be tested in order to prove or disprove the hypotheses. Section VI presents and analyzes the results. Finally, Section VII presents the research conclusions and suggested future work.

II. Background Research for Agent Based Network Security

2.1 Overview

Current network threats are various and quite serious, see Section 2.2. Accordingly, there are many contemporary approaches to network security. A recent trend is to decompose network security into multi-agent systems, in order to both distribute and automate security tasks, see Section 2.3. Multi-agent systems can be formalized in a variety of Ways 2.4. Self organization and tangled hierarchies are appropriate concepts for solving the problems of decentralized communication and control, see Section 2.5. To generate the self organized multi-agent system (SOMAS), search algorithms are necessary for finding optimum behavior configurations, see Section 2.6, and simulations for the simulating the generated system, see Section 2.7. Once the behaviors are generated, visualizations are required for viewing the results, see Section 2.8, and test and analysis procedures to determine whether the SOMAS goals are achieved, see Section 2.9. Finally, the possible languages for implementing the SOMAS framework are covered in Section 2.10. However, this chapter only provides background material for the design and implementation discussions. Those discussions are addressed in Chapters III and IV.

2.2 Network Security Overview

The realm of computer networks crosses a wide variety of disciplines. From the theoretical angle, graph theory is very important for determining crucial invariant network properties and developing efficient routing algorithms. The computer scientist also has a tremendous job of getting all the disparate networked systems to talk with each other, without also bringing the information flow to a standstill due to complexity overhead. Then, from the engineering perspective, computer networks are extremely complex. Not only must the engineer know how to route information around the network, he must also be knowledgeable about all the kinds of systems that can use and interact with the network. The great complexity inherent in computer networks means it is very difficult to plug all the holes and keep out threats. It seems that every day there is yet another debilitating attack vector. Thus, the network defenders has a very difficult task cut out for them of keeping the information flows safe and secure.

2.2.1 Network Graph Types. There are numerous kinds of graphs that can form the network topology. The simplest topologies are line, circle, star, and randomly interconnected, see Figure 19. Additionally, these graphs can possess certain characteristic. An example is planar graphs, where the graph can be created on a 2D surface without overlapping edges. Non planar graphs are the complement of planar graphs, encompasing graphs that require three dimensions. However, three dimensions is the maximum number of dimensions necessary to connect any graph without crossing edges. This is intuitive because if any edges cross in two dimensions they can just go around each other in the third dimension.

Scale invariant graphs consist of sections that exhibit the same pattern of edge degrees for any arbitrary section of any scale. While not exactly the same, often campus scale and larger networks exhibit this pattern, see Figure 16. Small world graphs possess the property that every node is linked within N edges, where N is small in magnitude. All of these graph related issues are covered in the mathematical discipline of graph theory.

Such network graph characteristics have an impact on the best rules for generating certain swarm behaviors. For instance, if the swarm is on a randomly interconnected graph, then, probabilistically speaking, the same rules should work generally well regardless of where an agent is on the graph. On the other hand, if the graph is composed of highly interconnected sections that are, in turn, sparsely connected, agents require different rules to have the same effect depending on the section they are in, and whether the behavior needs to span multiple sections.

2.2.2 Networking Protocols. Most networks in the real world make use of a protocol for relaying information. According to the end-to-end principle [108], system design accumulates too much overhead if every part of the system tries to account for all possible needs. Thus, similar to the design principle guiding agent oriented modelling, it is best to have particular services addressed by the part of the system that actually requires the services. Thus, protocols are developed, which are schema composed of the bare minimum level of information necessary to perform the required service.

For example, in order to transport information from point A to point B, it is generally only necessary to know the location of point B. Thus, the transport service only checks the part of its schema that tells it where B is, and leaves the rest of the information alone. The protocols used by networks are often stacked, since higher level services are based on lower level services. In the Internet, the OSI stack [8] is used. There are low level protocols for moving information from point A to B [4], middle level protocols for splitting messages into packets and routing them along a particular path [10], and higher level protocols for dealing with end user application services [2]. The use of the protocol stacks places an overhead on the network transportation, and can introduce new forms of errors. The protocols also provide useful information. However, if the agents in the multi agent system are to interpret all the various protocols used by all possible networks they may inhabit, the agents become too large for use. Consequently, *the end to end principle is followed, and the job of interpreting the protocol data into information is pushed to the host.* The agent only needs to know an interface for accessing this information.

2.2.3 Real World Networks. There are numerous real world networks. They range from extremely local networks, such as between a laptop computer and a bluetooth phone, to global networks, such as the Internet or the telecommunications network.

As mentioned in Section 2.2.1, the properties of networks tend to change as they become larger. At the local level, networks are fairly ergodic, their measurement does not change significantly depending on where they are measured. However, communication in such networks takes on too much overhead once the network passes a certain size. In this case, the network becomes split into sub-sections, which, in turn, contain other subsections. As the scale increases, the subsections resemble the global structure, whereas as the scale decreases the subsections resemble the local structure [86].

Networks can either be static or dynamic. Dynamic network topologies are most often found in wireless settings, since wireless nodes tend to change their geographical relationships. An example of an existing dynamic network topology is the Network on Wheels (NOW) [87]. NOW is an ad hoc network formed between cars on the highway. Utilizing the network, the cars can route themselves very effectively and efficiently by identifying existing routes, traffic patterns, and problems.

On the other hand, there are a greater number of static networks. These can be found ranging almost every institutional domain, from the financial sector, to the business world, to military operations. The networks particular to a specific domain, such as finances, tend to be fairly sparse with high bandwidth between nodes. Networks aimed at a more general demographic and use more hierarchical organizations to provide more efficient communication.

Once network topologies are implemented in the real world, new concerns come into play. For instance, the links between nodes take on important characteristics, such as bandwidth and latency. Furthermore, links can become hyperedges, as in the case of wireless networks, where the same medium links all network nodes.

Communication and the routing of information also becomes a concern. While Metcalfe states the addition of network nodes adds polynomial value to a network [81], Reed explains how the network actually provides more than just communication between nodes, it provides the basis for higher level networks to form. Today, one popular variant of this idea is social networking. The possibility of higher level networks raises the potential value of a new network node to exponential rates of increase [104]. However, this value is both negative and positive. Therefore, the net gain from an added node may actually be negative, and the increase in nodes can rapidly lead to destabilizing the network. The interplay between negative and positive gain led to a reevaluation of Metcalfe's and Reed's laws in [91], stating that the rate of increase, while still superlinear, is more on the order $O(n \log n)$. According to the authors, this growth rate more accurately reflects empirical observations.

2.2.4 Network Threats.

- *Web Based Insider Attacks:* According to the multitude of internet analysis and military leadership, we are fighting a losing battle against those who create malware. Although, the external worm threat is currently under control, the computer virus signature types have grown to astronomical numbers. While any unpatched computer that is hooked up to the internet is guaranteed to be attacked and compromised in less than 20 minutes [35], most attacks are ineffective against a properly protected computer. Consequently, the contemporary malware effort has moved to the client side and application layer, embedding exploits in web pages, emails, and external storage devices such as thumb drives [120]. The implication is that the threat has moved inside the network [80] [67]. Thus, border control is no longer adequate. Intrusion detection must look both at network traffic and host activity. Any kind of defensive system must be able to handle an internal threat by identifying it, quarantining it, and eliminating the malicious entities involved [93].
- *Botnets:* Recently, there have been many notable large scale botnets. Examples include Storm and Nugache [111], which some have estimated have over a million infected PCs. Such botnets are used for a wide variety of purposes, such as denial of service attacks, password cracking, information stealing, etc. Furthermore, these botnets are maintained by software engineering professionals, and are difficult to counteract [44].
- *Denial of Service Attacks:* Even though the threat of external infiltration by worms and viruses is low, denial of service attacks are still a very real problem, as

recent events in Estonia show [136]. The threat is heightened when the attacking computers can be within as well as outside the targeted network, brought about by internal intrusion.

- *Information Exploitation and Corruption:* The degradation of network performance results in the corruption and destruction of information. Besides corrupting information, malicious agents exploit and remove confidential information from the networks .
- *Counter Defense:* Finally, the defenders of all the information are the secondary, yet immediate, target of malicious agents; since defense keeps attackers from desired information.

2.2.5 Network Security Metrics. To gauge a network security system, it is important to have metrics for evaluating its performance. There are two aspects to network security metrics. First, the issue is what and where to use them. Second, the issue is how to measure the security metrics are.

2.2.5.1 Use of Security Metrics. There are many different areas in which security can be measured [21]. It can be measured at the network layer such as counting the number of packet types. It can be measured at the user level by monitoring network usage patterns. It can be measured at the organizational level by determining the level of security education. However, it is possible to have too many metrics, such that operators can suffer from data overload [54].

It is important to pick a minimal set of key metrics in order to make the job of security monitoring manageable. To do this a person needs to determine the network security areas that are important [106] and establish a method for creating a definitive metric [97]. For instance, a good list of security areas [106]:

- Policy and compliance
- Network and machine monitoring

- Outreach and education
- Legal compliance: DMCA, PCI, FERPA, etc.
- ID: authorization and authentication
- Asset protection
- Privacy

A good procedure for creating metrics [97]:

1. Define the metrics program goal(s) and objectives
2. Decide which metrics to generate
3. Develop strategies for generating the metrics
4. Establish benchmarks and targets
5. Determine how the metrics will be reported
6. Create an action plan and act on it
7. Establish a formal program review/refinement cycle

2.2.5.2 Measuring Security Metrics. Once the metrics have been established then the issue is how to measure them. Metrics can be measured with or without a model, for instance a simple model-less metric is to measure how many network incidents occur each month with different levels of severity. However, such metrics only specify how a system is currently doing, or may do in the future. In order to define the system's security characteristics a-temporally, the system must be formally modelled in a way that captures its general structure. The model itself is not necessarily measured, though it may, but measurements are taken as the model is simulated in different ways.

A number of researchers use different variants of POMDP models to do this, while other researchers have developed specialized formal models for network security. In one example, POMDP models have been used to characterize a system's survivability [32] by establishing policies to maximize the uptime for network elements. A similar work was done by [109]. The authors of [103] use POMDPs to define optimal routing protocols. In [141] the authors develop a model to counter multi-stage collusive attacks and develop two algorithms to find the key observations in the POMDP model that allow the most dangerous attack vectors. By removing these observations from the system, they mitigate the attacker's ability to successfully complete his attack.

Besides POMDP models, a there are a few other formal models that are used, which tend to more directly resemble a network or an attack. The authors of [79] create a model that accurately reflects a conventional network, down to the use of IP addresses. The authors of [94] use an attack graph, which is a graph that links different attack nodes to each other, to derive the weakest conditions for a successful network attack on a given network. Networks can thus be ranked against each other using these conditions such that a network with weaker conditions is more vulnerable.

2.3 Multi-Agent Based Network Defense

Due to the great complexity of network defense, it is a tall order for a single application to take care of. Just as one police officer cannot patrol a city and enforce the law, so one network defense solution cannot capture the totality of network defense. It makes sense to divide up the job of network defense so there are different applications for different concerns. Once the job of defense is divided up among different applications around the network, network defense is a distributed system, and requires a different way of thinking than centralized systems.

2.3.1 Distributed Systems. Middle-ware provides a unified mechanism and interface for access to system-wide services. The goal of middle-ware is to make the system invisible to the end user. This discipline of distributed systems is providing middle-ware that fulfills, as best as possible, the tradeoff between transparency, locality, and availability [127]. The end user should be under the illusion that they are only accessing local resources through a single system. An important manifestation of this design today is service oriented architecture [96]. Service oriented architecture is a middleware system that aims to provide access to a diverse set of resources such that the end user can combine them in arbitrary manners to fulfill a set of objectives.

There are a number of issues involved with providing the illusion of accessing a single, local system. First, a universal chronology is essential in many applications. One example is the make tool, which requires chronological information to know which files to compile. Second, the location of system wide resources must be known so they can be accessed. Finally, both issues have to deal with the possibility of failure and need to be able to backtrack so failures do not result in corruption of the system state.

One of the most important concepts for this discipline is the end-to-end principle [108]. The end-to-end principle states that concerns should be handled at the point where they are needed. For instance, if only some hosts in a network need secure communication, and there are varying security requirements, it is best for the particular hosts to handle their security instead of each node in the transport layer of the OSI model. The goal is to prevent bloat and fragility.

2.3.2 Agent Oriented Design. Agent oriented design can be considered an extension of object oriented design. Its distinguishing feature is that the objects are autonomous, instead of only reactive. That is, the objects do not depend on external triggers to act. Thus, there are two primary facets to agent oriented modelling: autonomy and systemic. The first facet is reflected in the domain of intelligent agents, which is covered in the following discussion. The second is found in the study of multi-agent systems, which is covered in Subsection 2.3.3.

The author of [62] makes the case for agent oriented modelling in complex problem domains characterized by stable sub components. The primary idea is that such large systems are decomposed into individual agents. This reduces the application logic because the system no longer needs to account for every combination of agents. Additionally, the system becomes more flexible, since new agents can be introduced and old ones removed without having to change the system architecture. Finally, since the agents are autonomous, they can carry out complex group tasks without external involvement [62].

The general capability to accomplish the tasks is called the swarm's behavior. A behavior can be composed of sub behaviors, which accomplish the necessary subtasks to complete the main task. Besides task completion, behaviors can also be general characteristics that most task completion requires, such as swarm stability. Nature can be modeled as a network of agents. An example of the agent based systems is an ecosystem. An ecosystem is composed of a great variety of entities all attempting to maximize their local reward. Yet, with local maximization they are able to create a superstructure conducive to all their goals. An agent is an information processing device. Since all physical agents are finite, they can only process a finite amount of information. Consequently, no agent has perfect information and cannot completely represent its environment.

However, the aggregate of communicating agents can be capable of representing their environment to a greater degree, and process this information as a group. Additionally, even though in theory a swarm is not computationally different than a single agent, real worlds constraints imply a swarm with decentralized information processing is inherently more capable than a single agent. Thus, in a swarm there is always an irreducible, emergent level that can be considered an information processing unit itself. This is the basis for the self organizing and multi-agent aspects of SOMA. The self organization is inherent in the emergent, irreducible ruleset.

Once a system has been decomposed into multiple agents, then the question becomes one of how to organize them. multi-agent systems (MAS) can either be centrally organized or decentralized. MASs can have a variety of different communication structures, ranging from flat, to hierarchical, to a hybrid of both. Finally, the MAS can either be composed of homogeneous or heterogeneous agents. A similar topic to multi-agent systems is multi-agent swarms. A swarm is a subtype of a MAS. Whereas a MAS can take on many different kinds of organizations and levels of autonomy, swarms tend to be flatly organized and highly autonomous. A common example of a swarm system is boids [105] by C. Reynolds.

Such multi-agent systems are covered extensively in the literature, both in theory and application. On the theoretical side, [90] describes a language and method for creating a multitude of virtual machines that evolve themselves, that are both sustain themselves and their environment (*autopoesis*) and accomplish this through stable feedback mechanisms (*hypercycles*). The author of [73] gives an overview of the state of the art in swarm intelligence in 2000. Swarm intelligence is an important augmentation to multi-agent design. Instead of modelling agent cognition explicitly, as Wooldridge and Jennings document as an unsuccessful approach in [61], cognition (intelligence) is implicit in the large number of simple agents. This is an elaboration on the ideas behind Brooks' subsumption architecture, non representational reasoning and intelligence [28].

The work of [95] is a more recent overview (2005) of cooperative multi-agent learning (similar to swarm intelligence). The authors state that the field of multi-agent learning suffers from lack of realism. The number of agents in a group is generally quite small, the environments are limited, and the objectives simplistic. On the more application oriented side [134] explains how swarm intelligence leads to effective route discovery in networks, [128] is a dissertation on the use of ant colonies to provide fault tolerance for network communication, and [72] combines the business version of multi-agent systems, services oriented architecture, with self organization.

2.3.3 Intelligent Agent Framework. As argued by Stytz, et al [126], military networks require a distributed intelligent agent framework for security in order to avoid the weaknesses associated with centralized control structures, such as lack of scalability, single points of failure, and fragility. According to Servat and Drogoul. [116], the networks of the future should be characterized by a ubiquity of mobile end devices, even perhaps with nanotechnology. Such a large, heterogeneous environment makes centralized control extremely difficult and costly, which further implies that controlling the system requires a MAS.

Intelligent agents are agents that have decision making abilities. These decision making abilities are based on artificial intelligence (AI). AI consists of knowledge representation and search [88]. Knowledge and rule representation in agents can be composed of first order propositional logic, modal logic, and probability [37].

There are a number of outstanding problems in AI that so far prohibit the creation of strong AI that emulates human intelligence, and thus truly intelligent agents. One significant problem is the framing problem, which is determining what information to retain from past states. The information from the past can increase quite rapidly, and it is difficult to predict what the future requires. Thus it is currently an unsolved problem. Another issue is dealing with uncertainty. It is possible to model uncertainty for a single agent and develop policies for acting with uncertainty for small problem domains. But, solving the general problem is PSPACE-complete and increasing the number of agents makes the problem even more difficult. All of these issues are covered in [88].

2.3.4 Mobile Agents. In [110], the authors argue that allowing mobility in software agents allows the MAS to have a greater range of ability. Yet, there is a danger of multi-agent systems devolving into chaos, and adding a new degree of freedom, with the allowance of movement, threatens to produce an extremely unstable system. Consequently, human involvement is required. In this regard, the concept of controlled self organization has been developed [27], essentially a hybrid between the extremes of completely autonomous self organization, and completely controlled external organization.

2.3.5 Container Model. Once agents are mobile it can become quite a complex task to allow them to operate in a heterogeneous environment, which is especially problematic since most networks are very heterogeneous. One way to solve this problem is to use a container model for the agents. Essentially, the container is a specialized virtual machine for agents. The container abstracts away the host specific details and provides the agent with a common interface of sensors and actuators across all hosts, applying the end to end principle [108]. Additionally, the container can provide utility functions for local agent interaction, such as providing each agent access to the data structures of other agents based on access rights. Formally, the container is characterized by the following tuple:

$$< Parameters, Accesslist, Sensors, Rules, Actuators >$$

Abstractly speaking, the agents are essentially sets of rules and parameters in the container which can be shifted from container to container. On the other hand, the container specific parameters and rules are not transported. An agent is differentiated from another agent by the rules that are transported and the access list.

2.3.6 CyberCraft. To defend against a network attack, a comprehensive solution is needed, allowing security to quickly be pushed out to all nodes and rapidly report back incidents. This security solution can be considered a form of middleware, a distributed computing system that all users of the network interact within. The Air Force is developing secure middleware called Cybercraft [65]. While the exact implementation design is still in development, it is based on a container model. Each node receives and deploys software "payloads" governed by a policy. Using these payloads, the Cybercraft architecture provides the operator with situational awareness.

With a CyberCraft infrastructure in place, it remains to populate the infrastructure with payloads. The payloads can generally be described as agents, and as such are encompassed by agent oriented design. Using the situational awareness capability, the ultimate aim of the CyberCraft model is to automatically respond to cyber attacks using the payloads. However, since the system is meant to be largely

Table 1: Taxonomy of commonly used MDP types

Type	Scale	S	O	A	T	Ω	R
MDP	L	L	N/a	L	L	L	L
DEC-MDP	G	L	N/a	L	L	L	G
POMDP	L	L	L	L	L	L	L
DEC-POMDP	G	G	L	L	R	L	G
I-POMDP	R	R	R	L	R	R	R
R-MTDP	G	R	L	R	R	L	G

L = Local, R = Regional, G = Global

autonomous, trust is the primary concern. The commanders have to be sure they can trust the CyberCraft to accomplish their intent without disrupting the system.

2.4 Agent Formal Modeling

There are many ways that agents can be formally modelled. The benefit of using a formal model, as opposed to just implementing an agent design in code, is the generality of the formal model. Consequently, anything that applies to the formal model applies to the particular agent implementation in question.

2.4.1 Markov Decision Processes. Markov decision processes (MDP) are reducible to the agent schema as outlined by Russel and Norvig in [88]. The agent schema consists of sensors, actuators, and state. The agent acts within an environment that may or may not have rewards. When the agent cannot obtain a full observation of its environment, it is represented with a partially observable Markov decision process (POMDP) [64].

The elements of the schema correspond to the POMDP tuple in this way: S is the agent's state, A are the agent's actuators, O are the agent's observations, and R are the rewards the agent can receive [88]. Along with the basic agent schema described in [88], there is a taxonomy of agents, ranging from purely reactive agents to decision theoretic agents. The mapping parameters T and Ω are the symbolic equivalent of the agent's cognitive capabilities. Whereas MDPs are a simple matter to solve and generate a policy from, since they are P-complete [88], POMDPs are a more complex matter because the true state has to be inferred from observations. POMDPs are consequently PSPACE-complete, which is only tractable for small problem domains. POMDP models can either be solved for an finite horizon, or for a infinite horizon [64]. The horizon denotes the number of transitions in a Markov chain a model is evaluated for. A finite horizon can be solved using straightforward numerical techniques, such as matrix multiplication combined with various pruning and reward propagation algorithms. The approach used can either be a feedforward or a backtracking algorithm. The backtracking approach is most convenient if a utility function exists. Generating a policy, then, is a matter of backtracking from the end states with rewards. If no utility function exists, then the feedforward algorithms give a total policy for all decision branches. In either case, the complexity of the decision space climbs exponentially as the number of transitions increase as with a breadth first search. The complexity is $O(b^t)$, where b is the branching factor, i.e. the number of decisions an agent has for any given state, and t is the number of transitions.

Solving for the infinite horizon is much more complex, and requires mathematical analysis of the model, since there are a countable infinite number of transitions to be evaluated using straightforward numerical techniques. One such infinite horizon technique is to find the Nash equilibrium of a competitive multi-agent model. The infinite horizon can be approximated by using finite horizon where the values have converged to within a certain threshold based on an error criteria.

A standard POMDP only represents one agent. However, it is often useful to model more than one agent such that all, some, or none share a reward function, yet all can affect each other and the environment. These models are the decentralized POMDP (DEC-POMDP), role based multi-agent team decision problem (R-MTDP), and interactive POMDP (I-POMDP), respectively. The DEC-POMDP and I-POMDP models encompass most of the other models. The exception to this is that they do not group actions into roles like the R-MTDP model. Additionally, while the majority of POMDP models are not recursive, and thus have countably infinite horizons, the interactive POMDP (I-POMDP) model is recursive, since the policy is based both on the observations of the environment and the beliefs the agents have about each others' policies [40]. There are thus two different horizon axes, one for actions in the environment and the other for recursive levels in mutual agent beliefs [48]. Additionally, since the beliefs contain policies as well and thus the horizon consists of an infinite number of infinite sets, the I-POMDP's horizon is uncountably infinite.

Finding a DEC-POMDP policy is NEXP-complete [22] (see Section 2.6.3 for an algorithmic complexity discussion) and finding an I-POMDP policy is harder or impossible due to the potentially uncountably infinite horizon. In the literature, all deterministic, optimal techniques for solving these MDP models do not scale past a few agents in a very simple problem domain. Consequently, due to the great complexity in solving the general case of POMDP models, heuristic based searches and memoization [51] are necessary. Sometimes the problem domain lends itself to a significant dimensionality and complexity reduction. In the case of the I-POMDP, if only the previous state's agent beliefs are necessary to predict the next state, then the problem is solvable too.

Using principal component analysis, Spaan and Spaan have reduced a real world POMDP model complexity such that it is tractable to generate its policy [122]. Another technique, as used by [31], reduces the dimension space by filtering only the most commonly used observations. Another technique is to use a stochastic process to generate an approximation to the optimal policy, such as with stochastic particle filtering [41]. These techniques have met with limited success, but not at the level that is required by the SOMAS problem domain.

Besides the problem of intractability, neither the DEC-POMDP or I-POMDP model allows for the production or deletion of agents. This can be simulated by *a priori* establishing the maximum number of possible agents, and then changing how they influence the agent swarm based on transition rules representing creation and deletion. However, the maximum possible number of agents is significantly larger than the number that is usually needed, so these models are composed of a lot of wasted space and become even more intractable. Finally, the maximum number of agents can be potentially limitless, and a finite model is incapable of handling an arbitrary infinite set. To solve these problems, POMDP models are augmented with functions because functions can produce a potentially unlimited number of values while still having a finite representation. Of course, this means the models can no longer be solved in the general case by an algorithm, since the models can be Turing complete [11].

The traditional, finite *POMDP models, where * stands for extensions such as DEC and I, are equivalent to regular grammars in a finite state machines [3], where strings are state/action histories. To augment the *POMDP model with a function, the state transition function, T, is replaced with a model transition function, T', as detailed in formula (1). **A** represents a non empty set of actions.

$$T' : *POMDP \times \mathbf{S} \times \mathbf{A} \rightarrow *POMDP \tag{1}$$

As long as the number of agents is constant or monotonically decreasing, then the multi-agent model is equivalent to a finite state machine with no pushdown stores. Once the number of agents is monotonically increasing without limit, then the multi-agent model is equivalent to a finite state machine with one pushdown store. Finally, if the number of agents can either decrease or increase without limit, then the multi-agent model is equivalent to a finite state machine with two pushdown stores. A finite state machine with two pushdown stores is equivalent to a Turing machine [77].

As a concrete example, the T' function can add or remove agent tuples from an I-POMDP model, as well as change the other dependent agent tuples accordingly. Thus, T' adds lengthening and shortening rules for manipulating the size of the *POMDP tuple. According to [55], the existence of lengthening and shortening rules in a formal system means it is potentially undecidable. This is because such a formal system can be equivalent to a Turing machine, and thus susceptible to the halting problem. Therefore, the $F(*POMDP)$ models are algorithmically unsolvable in the general case, though this does not preclude them being mathematically solvable in particular cases.

2.4.2 Agent Ruleset. As Norvig describes in [88], the agent can be abstractly considered a table matching observations to actions, which Norvig terms the agent's "external" characterization. The "internal" characterization of the agent is a program. There are many different general characterizations for the agent program. A representation that is used in much of agent learning is a policy. A policy maps observations to actions, similar to Norvig's "external" characterization of agents. A policy can either be optimal or suboptimal, fixed or changing. It can either be generated *a priori*, as is often the case with a POMDP model, or it can be generated online, such as in reinforcement learning. [88]

Rulesets are the most general kind of agent program, since a program is essentially a ruleset (which may modify itself). The basic distinction in rulesets is between rules that have decision functions and rules that constantly activate their actuators. Policies are a subset of rulesets, since a policy is a ruleset that use a simple numeric comparison for the decision function.

$$Policy \subsetneq Ruleset$$

However, while a policy's decision function is essentially a simple numerical comparison, in the general case a ruleset can use a very wide range of decision functions. Once the agent ruleset is defined, it is considered more or less effective based on how well it achieves a certain objective. The ruleset used to achieve the objective is considered the agent's behavior. Since a behavior is described in terms of what it accomplishes, the behavior also includes the objective. Thus, an agent's behavior is its ruleset and the objective the ruleset accomplishes.

2.5 Self Organization and Entangled Hierarchies

In order to develop MAS based security systems, it is necessary to create a scalable, flexible control structure. However, a scalable, flexible control structure tends to be an anachronism in computer science, as added flexibility and scaling leads to rapidly increasing overhead. Consequently, as evidenced by the literature, research is drawing inspiration from the many successful large scale, self organizing MASs in nature [27] [116] [114] [115].

For instance, many stochastic global search algorithms are inspired by such systems. Simulated annealing models how the annealing process causes the molecules to become organized in steel, making the steel less brittle. Evolutionary algorithms are based on the idea that the process of variation and selection in nature, whether by environment with natural selection or by animals as with the Baldwin effect, causes the genetic code to become more organized according to a fitness function. The ant colony algorithm copies the capability of foraging ants to find the most efficient path between a food source and their nest without overarching guidance.

To investigate such self organizing MASs three related, yet distinct, concepts are discussed in this section. They are: emergence, self organization, and entangled hierarchies. *Emergence* refers to things that are composed of parts, but are not reducible to the parts and their interaction. For example, a car is composed of a body, an engine, and four wheels. However, when we talk about cars, we use the term "car" as if it is something other than the parts it is composed of.

There are various ways phenomena can emerge. They can emerge by accident, by design, or by some mechanism. One form of mechanism based emergence is called *self organization*. While similar in certain regards, emergence and self organization are distinct concepts [137]. Emergence refers to a macroscopic (global) pattern being generated from microscopic (local) interactions. Self organization is when a system robustly sustains itself through internal local interactions and observations, without the need for external or global direction. However, given that both refer to a local dynamics to global property mapping, they are often conflated.

In a self organized system, the parts influence each other. This influence can proceed in a strictly hierarchical manner, where the higher levels influence all subsidiary levels unidirectionally. Alternatively, the influence can proceed bidirectionally. Besides bidirectional influence, the organization may be composed of many disparate and intermingled hierarchies. This complex form of organization, composed both of bidirectional influence and intermingled hierarchies, is called an *entangled hierarchy.* To incorporate an entangled hierarchy in a self organized system, the global level must have a concise enough Shannon coding [118] that the local level can interact with it, given its more limited memory space.

This is where emergence is important. A global pattern means that the global level can be encoded more concisely than just the aggregate of the system parts. For example, the word "example" on this paper or monitor is more concisely encoded as a series of ASCII characters than as aggregate of the atoms or pixels (i.e. bitmap image) in its composition. So, if global or regional behaviors in the self organized system are emergent, then the local elements can refer to and interact with them, within their memory limits.

2.5.1 Emerging Behavior is a Real Phenomenon. The idea of emergence is widespread in both modern day science and philosophy. In both disciplines, the interest is founded on the assumption that reality can be entirely based on physical processes, therefore every science is derivative of physics. However, the reductionism that this idea implies does not cohere with the view of many of the scientific disciplines, who consider certain phenomena in their disciplines as not being easily or even possibly reducible to basic physics. On first glance, this may seem to be logically invalid. If the sciences study physical objects, and all physical objects run by the laws of physics, then are not all objects just particles in motion, by definition? While this is a compelling argument, there is a rigorous theoretical basis for the idea of emergence based on Gödel's incompleteness theorem [25]. However, the relevance of this insight requires a proper understanding of reductionism.

First, it is important, as always, to define the terms being used. There are at least 3 distinct meanings of "emergence:"

1. It is practically intractible describe an "emergent" property in terms of its lower level structure. Notice that this says nothing about the *existence* of the emergent properties. The properties may or may not really exist in this case. For example, most people cannot describe a car in terms of its mechanisms, and instead describe it in terms of its function. However, it is perfectly possible to describe a car completely in terms of its mechanism without any loss of detail.

2. The "emergent" property has an existence distinct from its lower level structure. An example of such an emergent property is the color red. Even if everyone sees a different color when they see the wavelengths that produce the color red, we all (for the most part) know what each other are talking about when we say "red." This happens despite the fact that no one's neuro-chemical process is composed of the same particles and the light's photon energies and wavelengths that produce the vision are all completely different.

   Another example is DNA. The specific strand is made up of a unique set of molecules and cells. However, the information it encodes is universal. The same configuration means the same thing regardless of the particular material it is made of. Thus, the information is distinct from the material and is an "emergent" property. However, while this idea may be philosophically compelling (though it is still quite controversial in the philosophical disciplines), it does not find easy acceptance on rigorous grounds among the sciences.

3. A property cannot be produced by lower level structures that lack information about the property, even though its existence is reducible to lower level structures. Kolmogrov complexity [71] is one quantification of this idea, that there are certain programs (given a computational model) that cannot be re-

duced to still smaller programs. A program's Kolmogrov complexity is the smallest program necessary for its reproduction. A familiarity with Godel's incompleteness theorem shows Kolmogrov complexity is a particular instantiation of Godel's theorem. Then, given the assumption that reality is computational, which seems reasonable if reality is quantized and it all can be emulated on a computer of adequate fidelity, irreducible physical phenomena is at least plausible. The benefit of definition 3 for emergence is that it does not contradict any of the assumptions of modern science, while still giving a theoretical justification for properties that are semantically irreducible, instead of merely syntactically irreducible as in the case of definition 1.

Thus, with definition 3 in mind, it is possible to answer the original question "isn't the concept of emergence meaningless since everything is reducible to physics?" with a "no." Formulæ (2) and (3) describe the modality of the bidirectional relationship between emergent phenomena and physical law in the more general terms of modal logic.

$$\text{emergent phenomena} \rightarrow \Box\text{axiomatic system} \tag{2}$$

$$\text{axiomatic system} \not\rightarrow \Box\text{emergent phenomena} \tag{3}$$

Emergent phenomena necessarily imply an axiomatic system, such as the laws of physics, but the axiomatic system does not necessarily imply a particular emergent phenomena. The emergent phenomena are, in turn, specified by information that is latent in the physical system, analogous to the Kolmogorov complexity of a computational representation. *In order for a decentralized MAS to remain organized without resorting to centralized organization, its behavior must be emergent.* Therefore, the theory of emergence is essential for the design of a decentralized MAS.

2.5.2 Self Organization. Self organization is the essential mechanism for producing robust emergent behavior in a MAS. A pattern is emergent if it is not explicitly specified in either the swarm or the environment, and it is unlikely to happen given a random set of parameters. External organization describes how a MAS behavior necessarily emerges primarily due to environmental feedbock. Internal organization, or self organization, describes how the behavior in the MAS necessarily emerges primarily due to internal feedback.

Since internal organization does not depend on the environment, it is more robust than external organization. Hybrid organization, of course, describes how both sources of feedback provide equivalent organization for the swarm. However, in all cases, no new information is being created. The information necessary for the behavior to emerge is latent within the system, whether MAS, environment, or both, in some way.

2.5.3 Qualifying Self Organization. To construct a self organizing swarm, it is necessary to qualify when a swarm is self organized. If the information necessary for a certain emergent property is latent in a system, then the question becomes how does the emergent property emerge? A new property implies new information. Yet, information is said to be conserved [38], and new information must come from somewhere.

There are a number of ways patterns can be constructed. The patterns can be explicitly constructed by a designer, such as a construction chief at a building site. Alternatively, the pattern can be the result of an environmental template, such as a river bed. However, in each of these cases, we do not say that the pattern is emergent, because the pattern already exists.

In nature, there are occurrences of what are called self organizing systems. These systems show a pattern of behavior over a long time despite the fact that there is no overarching control of the system. For instance, ants and termites display this kind of behavior to an amazing degree. Even though many human organizations rely on hierarchical control, there are also organizations that operate without such control, though this is rarer than in the insect world. The benefit of self organizing systems is that they can easily adapt locally to changing environments while maintaining a global objective. This is a very useful idea for mobile agents in a network since networks can have a wide variety of configurations and change rapidly.

There are a number of definitions of self organization. They range from the observable, such as that of [53] who states that self organization is the emergence of a global, organized pattern in a system without any overarching control, through local observations and interactions; to the measurable, such as the notion that the statistical complexity of the system increases faster than the statistically complex information flows into the system [101].

With these two definitions, there are two primary ways self organization can be qualified: in terms of *observable system dynamics (qualities)* and in terms of a *metric (quantities).* The first way is to identify whether the global pattern results only from local observations and interactions. The second is to measure the complex information flows. In the latter case, a quantitative metric of self organization can be constructed and used as an algorithm heuristic. With the observational qualification of self organization, it is possible to visually identify, and thus humanly engineer a self organized MAS.

It is not always easy to formalize the qualities we observe, so the observational qualification provides the most general method of qualifying self organization. In [114] the authors present an extensive list of observational qualities that depict a self organized system. With the self organization metric, a search algorithm can find a self organized MAS automatically. Of course, it is essential that the metric specifies a swarm is self organized when it can be observed to be self organized.

The modal, unidirectional relation between the two qualification methods is specified in propositional logic form in formulæ(4) and (5).

$$\text{metric qualification} \rightarrow \Box\text{observational qualification} \tag{4}$$

$$\text{observational qualification} \not\rightarrow \Box\text{metric qualification} \tag{5}$$

The successful evaluation of the metric always implies self organization is observable, but observed self organization does not necessarily imply the metric measures the self organization. Nor does it even imply a metric is possible.

2.5.4 Quantifying Self Organization. The organizational characteristics of the swarm can be formalized described in reference to transition function $\mathcal{T}$ (6). Θ is the set of swarm states. Σ is the set of environment states. Ψ_Λ is a set of state classes, Ψ_λ, where each class is composed of $< \Theta_\mathcal{O}, \Sigma_\mathcal{O} >$ tuples. The subscript $\mathcal{I}$ means input and the subscript $\mathcal{O}$ mean output. A subset, $\Psi_{\Lambda\times\mathcal{B}}$, of Ψ_Λ maps to a set of behavior classes according to function $\mathcal{M}_\mathcal{B}$ (7).

$$\mathcal{T} : \Theta_\mathcal{I} \times \Sigma_\mathcal{I} \rightarrow \Psi_\Lambda \tag{6}$$

$$\mathcal{M}_\mathcal{B} : \Psi_{\Lambda\times\mathcal{B}} \rightarrow \mathcal{B} \tag{7}$$

A behavior b in $\mathcal{B}$ is emergent if predicate $\mathcal{E}(b)$ (8) is true, and it is decentralized. Altogether, the formula states if b contains a set of state classes that are unlikely to be obtained given all possible transition functions, then b is emergent. Additionally, this result can be generalized to a subset of $\mathcal{B}$ by considering the subset to a particular behavior.

$$\mathcal{E}(b) \rightarrow \exists\psi_\lambda\exists\Psi_\lambda\exists\Psi_\Lambda^\mathcal{E}\{\psi_\lambda \in \Psi_\lambda \wedge \Psi_\lambda \in \Psi_\Lambda^\mathcal{E} : \mathcal{M}_\mathcal{B}(\Psi_\Lambda^\mathcal{E}, b) \wedge \mathcal{P}_{\mathcal{T}*}(\Psi_\Lambda^\mathcal{E}) < \epsilon\} \tag{8}$$

The function $\mathcal{P}_{\mathcal{T}*}$ is a probability function based on all transition functions $\mathcal{T}*$ that computes $(\Psi_{\bar{\Lambda}}|\mathcal{T}*)$ where $\Psi_{\bar{\Lambda}}$ is some subset of Ψ_Λ. ϵ is a threshold that is at most 0.5. The is based on Shannon's information theory, where the amount of information in a behavior is the $\log_2$ of the inverse of its likelihood of occurring, or $\log_2(\frac{1}{\mathcal{P}_{\mathcal{T}*}(\Psi_\Lambda^b)})$. If the information content of a behavior is normalized according to the information in the system as a whole, consisting of the set $\mathcal{T}*$, then any result more than unity implies the behavior contains more information than the system can account for. If

$\mathcal{P}_{\mathcal{T}*}(\Psi^b_\Lambda) < 0.5$ then the information content must be more than 1. Thus, it fulfills the condition of "unlikely to happen given a random set of parameters."

The other two criteria of local observation and interaction are grouped under the qualification of being decentralized. These, in turn, can be measured by another metric that is introduced in Section 2.5.6, which discusses entangled hierarchies. To talk about self organization, then, is a matter of examining the relation between the sets $\Theta_{\mathcal{I}} \times \Sigma_{\mathcal{I}}$ and Ψ_Λ, and functions $\mathcal{T}$ and $\mathcal{M}_{\mathcal{B}}$. If sets $\Theta^\omega_{\mathcal{I}}$ and $\Sigma^\omega_{\mathcal{I}}$ fulfill formula (9), then $\frac{|\Theta^\omega_{\mathcal{I}}|}{|\Theta_{\mathcal{I}}|} \propto \frac{1}{\mathcal{SO}_m}$ and $\frac{|\Sigma^\omega_{\mathcal{I}}|}{|\Sigma_{\mathcal{I}}|} \propto \frac{1}{\mathcal{EO}_m}$. If $\frac{\mathcal{SO}_m}{\mathcal{EO}_m} > \mathcal{EO}_\epsilon$ then the behavior is externally organized. If $\frac{\mathcal{EO}_m}{\mathcal{SO}_m} > \mathcal{SO}_\epsilon$ then the behavior is self organized. $\mathcal{EO}_m$ and $\mathcal{SO}_m$ are thresholds for their respective properties.

$$\forall \theta_{\mathcal{I}}, \sigma_{\mathcal{I}} \{\theta_{\mathcal{I}} \in \Theta^\omega_{\mathcal{I}} \wedge \sigma_{\mathcal{I}} \in \Sigma^\omega_{\mathcal{I}} : \theta_{\mathcal{I}}, \sigma_{\mathcal{I}} \in \mathcal{T}(\Psi^b_\Lambda) \wedge \mathcal{E}(b)\} \tag{9}$$

Many scientists have developed methods of creating or measuring a particular characteristic of self organization. For instance, a set of robots that group together can be said to show self organization [18]. A decentralized intrusion detection system that uses two agent populations to detect and eliminate intrusions relies upon self organization for the populations to work together without centralized control [45]. Robots that evolve specific, global behaviors are self organizing as well [18]. An evolving modular robot that reduces the entropy in its behavior also exhibits self organization [102].

Other researchers have developed more general metrics for self organization that can be applied to a general class of problems, or ways of characterizing the conditions for self organization. [29] lists and relates the general characteristics of self organization. [140] describes a set of conditions that lead to emergence, one essential characteristic of self organization. A few general quantifications of self organization have been developed by [34, 117]. The authors of [101] presents a good overview of different self organization quantification techniques applied to selected problems. Out of the quantitative techniques, the predictive information metric is the most general of the set, and is fairly straightforward to implement. The essential idea is that as the number of chronological sequences (l^-) that can be used to predict future sequences (l^+) in a history increases, then the system is developing more structure, and hence becoming more organized.

$$I[x;y] = P(x,y)\log_2 \frac{P(x,y)}{P(x)P(y)} \tag{10}$$

Once a system's history is obtained, the sets L^- and L^+ are extracted and $I[l^+;l^-]$ is calculated for each element, resulting in a measurement of systemic organization. If the metric is used on sub sequences of the history, then the increase in organization can be measured.

It is important to note that the metric is opposed to both random and trivial histories. If a history is random, that means any given l^- and l^+ are equally likely to occur together and $P(x,y) = P(x)P(y)$. On the other hand, if a history is trivial, then the sets L^- and L^+ are minimal, again resulting in $P(x,y) = P(x)P(y)$. As long as $P(x,y) = P(x)P(y)$ then Equation (10) produces a zero value. Thus, once the metric gives a positive reading, the history contains organization.

A causal state is a probability distribution over future sequences. For each causal state there exists a set of pasts that each have the same causal state. With the use of the ϵ statistics, which is the minimal sufficient statistic for a causal state [117], Equation (10) can be simplified to Equation (11). This Equation states that the amount of predictive information is, per Shannon's information theory [118], the $\log_2$ of the number of causal states.

$$I[L^+;L^-] = \log_2 |\epsilon| \tag{11}$$

While the metric measures the amount of organization, this still does not specify whether the organization is created, emergent, or self organized. In the case of

SOMAS, emergence can be assumed, since an organization is not pre-built into the swarm and the swarm generates its behaviors through local observation and interaction. Whether the emergent structure is self organized or not depends on the amount of interaction within the swarm, which can be characterized by whether it exhibits an entangled hierarchy.

2.5.5 Constructing Self Organization.

Besides qualifying whether a given system is self organized, it is possible to create systems that exhibit self organization through certain rulesets. These rulesets tend to be developed through observing existing self organized systems in nature. For example, a well accepted situation where we say a pattern is emergent is in the case of foraging ants. The ants execute very simple rules only with reference to their local situation, yet they are able to accomplish global behaviors such as foraging and routing.

Within such a system there are *3 main principles at work: positive feedback, negative feedback, and information transfer.* For example, the positive feedback is the path reinforcement when ants relay new pheromones on existing pheromones when a good path is discovered. The negative feedback does not really exist in the ant system, except in pheromone decay and random exploration, but an example of such feedback is in the stock market where too many investors in a stock dilutes its value, eventually leading to a decrease in investors. Finally, information transfer is apparent in the ant system's use of pheromones. These three elements of negative and positive feedback, and information transfer, are generalized from a wide variety of self organized biological systems in [29].

2.5.6 Entangled Hierarchy.

In multi-agent systems, both human, mechanical, and computational, effective behavior depends on using both global and local information. However, usually the individual agents cannot observe all the information they need. Consequently, the information must be communicated from those that have it to those that need it. With small networks, this can be achieved effectively and efficiently by all to all broadcast, multicast, or pointcast. However, as networks grow in size, such communication schemes grow in overhead at an exponential rate. Even the number of hierarchies itself has an upper limit. Humans can generally handle between 3 and 7 unique items in their mind at once. Any more than this and they'll start becoming confused and making mistakes. Thus, an increase in hierarchy generates its own overhead, and beyond about 3 or 4 levels the point of diminishing returns is reached.

Consequently, there is an effective limit on the size a hierarchical structure can grow, and thus the size of a network. Of course, computers can process much more data in a given period of time than humans, so the hierarchical limit of computational networks is many magnitudes greater. But, there is also a limit to the size of computational networks that are based on hierarchies. Some use a more flattened system to avoid the inefficiencies of hierarchies. Beyond a certain size, the individual agents cannot establish a good global perspective, and such systems become direction-less. Alternatively, sometimes hierarchies can overlap, and two global perspectives combine. This can work if the perspectives are compatible, but often their goals conflict, again leading to lack of direction at the lower layers.

The communication problem can be characterized by routing the right information to the right agent at the right time. There are many different methods of routing this information, some more or less efficient. The goal is to produce the most efficient routing, and as such the problem becomes a minimization problem. Using a hierarchy is one effective heuristic for creating an efficient routing scheme. But, it is inherently inefficient in certain ways since it predetermines that communication with at least half the network proceeds indirectly. Unless communication structures fit such a structure exactly, the indirection always leads to inefficiency.

A second, more general inefficiency with communication, which does not depend on whether it is hierarchical or not, is the use of information packets. If the packets are only sent point to point, then either certain nodes do not receive all the information they need directly, or the sender must do a lot of bookkeeping to ensure all recipients get exactly what they need. Additionally, the latter case presupposes the sender has an adequate global view to know what information each recipient needs. Neither is this approach easily scalable. Yet, if multicast or broadcast is used, then often recipients receive redundant information. Essentially, the communication and information structure needs to be an emergent property of the communication itself, since any kind of external control presupposes a certain communication routing.

If the measure of a resources is the cost of its use, then the pairing of tuples to resources results in the resources' costs being respectively scaled according to their tuples' commonality in the actual communication structure. Summing the total resource cost for the routing scheme gives a measure of its optimality. The optimal routing scheme minimizes the resource cost. A symbolic formalization of this discussion is found in Formulæ (12-14). Where Ψ is a set of tuples relating route and information. $< \text{Route}, \text{Info} >$, $\mathcal{P}$ is a set of probabilities, abiding by the standard properties of probability sets. $\mathcal{R}$ is a set of resources. $\mathcal{C}$ is a set of costs.

$$p : \Psi \rightarrow \mathcal{P} \tag{12}$$

$$s : \Psi \times \mathcal{R} \rightarrow \mathcal{C} \tag{13}$$

$$m : \Psi \times \mathcal{R} \rightarrow \{0, 1\} \tag{14}$$

Formulæ (15-16) detail the nature of efficiently coded and inefficiently coded communication. For a given m_α, t_e is the threshold of $\frac{|M_\geq|}{|M|}$ above which m_α is considered an efficient coding and $t_{\not e}$ is the threshold of inefficient coding.

$$c(m) = \sum_{\psi \in \Psi} \sum_{r \in \mathcal{R}} m(\psi, r) \times p(\psi) \times s(\psi, r) \tag{15}$$

$$\forall m \{ m \in M_\geq \leftrightarrow m \in M \wedge c(m) \geq c(m_\alpha) \} \tag{16}$$

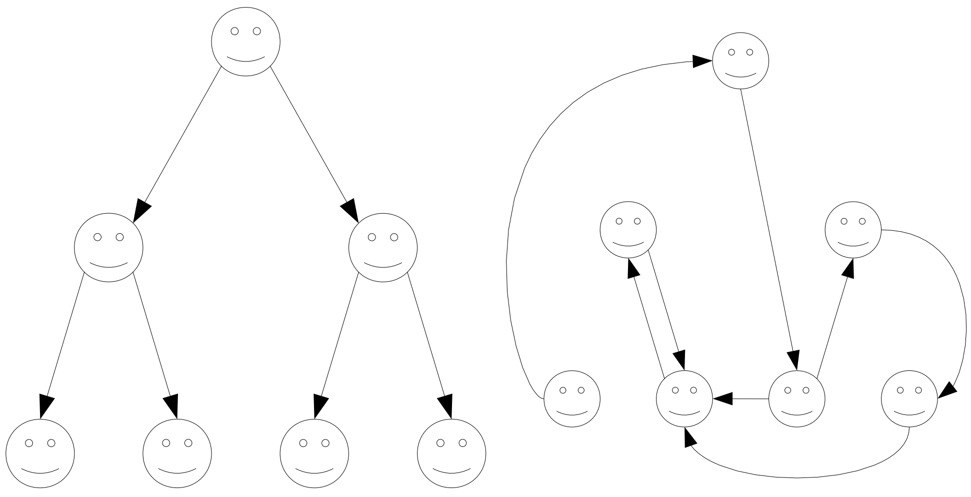
In general, the efficient communication scheme is called an entangled hierarchy. This is because the information sharing covers the whole range of pointcast, multicast, and broadcast, but not according to any general principle, only according to the actual information needs. Additionally, the data gathering and decision making communication does not proceed in only one direction, but both types of communication can proceed in both directions between any given recipient and sender. Thus, the communication routing structure can range from being completely flat to being a strict hierarchy. The optimal structure likely falls midway between the extremes, exhibiting a number of different interrelated hierarchies.

The stock market is a concrete example of both the phenomena of an entangled hierarchy, and its attendant difficulties. The stock market is an emergent aspect of worldwide trading. At the same time, investors interacting with the stock market influences the individual companies that are engaged in this trading. Thus, both the highest level in this hierarchy emerges from and is dependent on the lowest levels, while it also influences the lowest levels. However, if not pursued with care, an entangled hierarchy can become unstable.

An entangled hierarchy supported by the trusted system of CyberCraft provides the crucial resilient flexibility necessary to adapt to the constant changes of Cyberspace. At the same time, the example of the stock market shows the promise of entangled hierarchies must be met with caution. A communication structure is not self aware, and there must always be a human in the loop to help ensure stability.

Entangled Hierarchy vs Power Law Hierarchy: The internet is built based on a power law distribution [119], such that the number of hops between any two arbitrary nodes scales logarithmically with the number of network nodes. This is a very useful network design if the nodes' dependability increases with their link degree, and if the network topology is fairly static. However, if this characteristic cannot be guaranteed, then a communication structure based on the power law distribution is not necessarily beneficial. In such a case, the more general concept of a specialized entangled hierarchy may work better.

Figure 1: Normal hierarchy vs entangled hierarchy



2.5.7 Quantifying Entangledness. While self organization means that the swarm develops a consistent structure of behavior, an entangled hierarchy means that the structure exhibits hierarchical relationships between the different parts (such that certain parts exert more influence over the swarm's behavior than that of the influenced parts). The causal relationships do not only procceed in one direction in the hierarchy. An example comparing a normal hierarchy to an entangled hierarchy is shown in Figure 1.

The benefit of an entangled hierarchy is that it is a higher level abstraction that encompasses all the standard communication structures. There are two extremes to a communication structure: completely flat or completely hierarchical. In turn, there are 2 metrics that distinguish these extremes. They are influence ordering and causal relationship. An influence ordering means that there is a ordering of more influential to less infuential amongst the parts being measured. The causal relationship shows the direction of influence. In a hierarchy, the parts have a strong influence ordering and unidirectional causal relationships. Whereas, a flat communication structure has a weak influence ordering and bidirectional causal relationships. However, in an entangled hierarchy, the parts both have a strong influence ordering as well as bicausal relationships.

The entangled hierarchy can be measured as a multiobjective problem, detailed in the following equations. While these objectives do not exhaust the concept of an entangled hierarchy, they imply an entangled hierarchy.

$$\min(\text{argmax}_{rank}(|rank|)) \tag{17}$$

$$\max(\text{argmin}_{v\in\mathcal{V}}(|\text{v's unique cycles}|)) \tag{18}$$

Where *rank* is a given influence rank. $\mathcal{C}$ is a set of cycles. $\mathcal{V}$ is the set of vertices in the influence graph.

Minimizing equation (17) creates influence stratification amongst the vertices of the influence graph. In equation (18), a unique cycle is a cycle that v is in such that there is another cycle v is in where none of the other vertices in the second cycle are in the first cycle. Maximising this objective creates complex bidirectional causality.

2.5.8 Intelligence Augmentation. Intelligence augmentation (IA) is the study of the mutual interaction between man and machine for the ultimate benefit of man's intelligence. As exhibited by arguments such as Godel's incompleteness theorem [74], Searle's Chinese room argument [112], qualia [130], and other theoretical concepts, it is not very likely that true AI is logically possible.

However, these same arguments provide an important insight into the more useful direction for computer research, which is intelligence augmentation. Godel's work, in particular, highlights the specific ability that humans have, which is self formalization. This means that humans can identify a process they are performing and then formalize its syntax. Since all computer programs are state machines, they cannot represent themselves. Therefore, computer programs cannot self formalize like humans can. At the same time, computers are very good at running state machines, and humans are not. This suggests that the best interaction between computers and humans should allow humans to easily define a state machine for a given problem,

Table 2: Information source mapped to search

	Local Solution Space	Global Solution Space
Local Population	Deterministic	EA
Global Population	EA	EA

which the computer would then maximize and feed the results back to the human as a new problem.

Similar to our interaction with our body and the environment, this process gives rise to emergent properties, providing the human with new directions for investigation. This interaction between human and computer is called intelligence augmentation, since a human is using a computer to augment their intellect instead of as a replacement for their intellect.

Self organization is incorporated into the discipline of IA through the notion of controlled self organization [27]. The essential idea is to use self organization to give humans control of the system. This is one possible way to deal with the issue of instability in self organized systems.

2.6 Search Algorithm

A better search for a domain encodes more information relevant to the search domain. There are two sources this information can come from: before the search (offline) and during the search (online). During the search, there are also two sources of information: the list of partial solutions and the solution space. These sources of information can either be local or global.

The rating of a source before the search is based on the calculation of expected active information $P(T \in S|I_E) * I_A$. $P(T \in S)$ is the probability of the target being in the search space given I_E, the exogenous information. I_A is the active information [38]. The online sources of information can be combined with both the offline sources and other online sources. Table 2 describes how the source of information and the locality of the solution space that can be sampled specifies the kind of search algorithm.

The neighborhood to solution hyperarch denotes the search space and the offline source places a probability distribution on the search space. In a search operation, characterized as the modus polens rule $\exists P \exists Q\{P \rightarrow Q : P \in S_{Part}, Q \in S_{Srch}\}$, the selected partial solutions are the antecedent P and the selected solution space solutions are Q. S_{Part} and S_{Srch} can both be populated by searches as well, nested to an arbitrary depth.

The distinction between deterministic and stochastic search is the straightforward observation that given $P \rightarrow Q \vdash Q \in Q'$, $Q = Q'$ in the case of deterministic search and $Q \subset Q'$ in the case of stochastic search. For an arbitrary domain, the baseline search is the Monte Carlo sampling [138]. This is the baseline since any solution is just as likely to be picked as any other solution. Any search that performs statistically worse or better than the blind search has information about the search domain, regardless of whether that information is accurate or inaccurate [38]. The likelihood that an informed search finds a solution compared with the random search's likelihood provides a precise quantification of the information the search encodes regarding the search domain. This quantification allows an accurate (though not necessarily precise), and implementation invariant, comparison between search algorithms.

There are a large number of ways of formalizing search domains. The algorithm chosen is primarily dependent on the complexity of the search domain. If the solution can be decomposed into a partial solution search, then a search based on local information is likely appropriate. This method of searching is generally used for solutions that can be found in polynomial time.

On the other hand, in very complex search domains, there are many local optima, and both global and local information is needed to find the optimal solution. Additionally, in large search domains, it is usually necessary to use an approximation search instead of an exact search. Approximation searches are good at finding solutions in polynomial time, and they often provide good enough approximations.

Table 3: Algorithms mapped to fitness landscape characteristics

	Hilly	Craggy
Smooth	Tabu search	Evolutionary strategies
Rough	Simulated annealing	Evolutionary algorithm

Approximation searches are either deterministic or stochastic. The benefit of stochastic search, such as simulated annealing or EAs, is that it avoids getting trapped in local optima without the need for storing previously visited solutions. Due to the law of large numbers [5], if the probability the stochastic search algorithm finds the optimal solution is greater than zero, then it finds the optimal solution as the number of sampled solutions approaches infinity. To achieve the same result, deterministic searches must store their entire search history in some form in order to guarantee that the optimal solution is found. For a detailed description of a number of stochastic global search algorithms please see [82].

Selecting a good global search algorithm requires information about the fitness landscape. Such knowledge can come in a general form, such as knowing how rugged the landscape and how smooth the slopes are. For example, the smoother the slopes are and the fewer the number of slopes, then the more useful a hill climbing algorithm, i.e. local information search, is. A very rugged landscape can trap searches in local optima, so such landscapes should be searched using some form of global solution sampling [82].

2.6.1 Fitness Landscape Characteristics. Mountainous means the landscape has a lot of optima that vary greatly from the optimal in magnitude. Hilly means there are not many optima that vary greatly in magnitude. Smooth means the slopes that lead to the optima can be climbed fairly easily by a backtracking hill climber with a small amount of memory. Rough means a large amount of memory is necessary to find the optima.

2.6.2 Evolutionary Algorithm Packages.

- *GALib [132]:* GALib is a genetic algorithm library written in C. It is quite computationally fast, and is fairly popular. However, since it is written in C, its design is not loosely coupled, and can be unwieldy. It does not have support for parallelization. A major disadvantage of the package is that it is only a single objective genetic algorithm framework. The SOMAS domain is best expressed as a multi-objective problem. To evolve solutions in the SOMAS problem domain using GALib, it is necessary to use some kind of scalarization of the objective vector. This likely loses too much important information.

- *Swarmfare [100]:* The existing evolutionary algorithms are good, such as the self organizing genetic algorithm that uses entropy to perform differential evolution. The framework does not provide a lot of flexibility for the implementation of new algorithms. Specifically, much of the problem domain information is hardcoded into the algorithm, and it requires a significant amount of software engineering to make it loosely coupled enough to evolve solutions for other problem domains. For instance, the chromosome and gene sizes are a set length. The chromosome is directly tied to the phenotype level, which is the UAV platform.

  Swarmfare does have an inbuilt parallelization capability. It is a farming model that uses a master slave structure. The slaves are initialized on all the nodes and then request functions to evaluate from the master. Swarmfare also has an extensive GUI. The GUI makes it simple to understand how to use the framework, reducing the learning curve. The scenarios are configured by using external text files, so many different scenarios can easily be generated and tested. The simulator comes with visualization capabilities. The simulation elements consist of UAVs, obstacles (lines), and enemy targets. The UAVs can be programmed with a variety of behaviors.

- *ECJ [76]:* ECJ is a widely used general evolutionary computation package written in Java. It has been improved over a great many years, which comes with advantages and disadvantages. The main advantage is that the package has been software engineered extensively so it is fairly straightforward to develop complex algorithmic workflows. However, since the package has existed for so many years, there are legacy aspects that convolute the code, but are kept to preserve backwards compatibility.

There are a wide variety of evolutionary algorithm types that are already supported. All the standard representations and operators for genetic algorithms, genetic programming, and evolutionary strategies are contained in the package, such as bit strings, real vectors, and programmatic trees. Additionally, the package design pattern is based on the concept of Linux pipes, such that processing elements send their output to other processing elements. With these pieces, complex algorithmic graphs can be built.

ECJ is built with parallelization in mind, and can be parallelized using a number of different job distribution techniques. It can handle both farming and island approaches. That is, either distributing only the fitness function evaluation or also the solution variation and selection.

- *Open BEAGLE [47]:* Open Beagle is a evolutionary computation framework written in C++. Its object oriented design means, as with ECJ, it is easy to extend. Additionally, the framework can handle multi-objective evolution. Open Beagle contains the standard representations and operators for genetic algorithms and evolutionary strategies. Once the user specifies the objective function and the chromosome type minimal code is required to combine the rest of the package elements into an evolutionary algorithm.

  Open Beagle has a distributed computing library called Distributed BEAGLE. However, it is not actively developed. The latest release is 2004.

- *GEATbx [99]:* GEATbx is an extensive EA library written in MATLAB. MATLAB is not object oriented, so the design of GEATbx is not as easily extensible as Open BEAGLE and ECJ. Since it is in MATLAB, it has the most useful analysis and visualization toolset out of all the researched libraries. MATLAB can also run its code distributed on a grid, so GEATbx can be parallelized.

GEATbx has two significant disadvantages. The first is that the fact MATLAB is closed source means it is hard to have MATLAB programs interoperate with external programs. If all development and testing is to be done within MATLAB, this is not such a big problem. But, since MATLAB does not provide a package that meets the simulation needs, none MATLAB programs are a necessity. The second disadvantage is GEATbx has been commercialized and is now closed source. While the interfaces it provides are adequate for most needs, it is risky to assume they meet all needs. Waiting for the library to be extended, if it is, can cost a lot of time.

2.6.3 Algorithmic Complexity. There are two aspects to algorithmic complexity: time and space. An algorithm's complexity can be measured according to the amount of each it takes. Algorithm classes become progressively more complex, as shown by the following set relations [9]

$$P \subseteq NP \subseteq PSPACE \subseteq EXPTIME \subseteq EXPSPACE$$

These classes are defined as:

- P $= \bigcup_{k\in\mathbb{N}} DTIME\left(n^k\right)$
- NP $= \bigcup_{k\in\mathbb{N}} NTIME\left(n^k\right)$
- PSPACE $= \bigcup_{k\in\mathbb{N}} SPACE\left(n^k\right)$
- EXPTIME $= \bigcup_{k\in\mathbb{N}} DTIME\left(2^{n^k}\right)$
- EXPSPACE $= \bigcup_{k\in\mathbb{N}} SPACE\left(n^k\right)$

Where $DTIME$ signifies the amount of time to solve using a deterministic Turing machine. $NTIME$ signifies the amount of time to solve using a non deterministic Turing machine. $SPACE$ signifies the amount of space used, regardless of whether the Turing machine is deterministic or non deterministic, since the difference does not affect the power of the complexity class.

2.7 Simulation Packages

2.7.1 Network. The network simulation requirements for the SOMAS framework are very simple. Essentially, the simulator needs to contain nodes and edges. The impact of issues such as protocol stacks and bandwidth and latency are not dealt with in this study.

- *OpNet [7]:* OpNet is a widely used commercial package for network simulation work. It is used for many production applications and provides the greatest realism out of all the packages. However, since it is commercial, modification to the simulation is very limited. Additionally, it does not provide host level simulation. It only simulates the traffic flows between hosts. Consequently, it is not applicable for the SOMAS simulation since the simulation requires running new code on the hosts. It may be used in tandem with another simulation to provide realistic traffic conditions.
- *NS2 [6]:* NS2 is used to simulate networks. The ruleset and basic framework are hardcoded in C++, and the runtime behavior is manipulated by Tcl scripts. One the benefit of using NS2 is the scalability of the model's realism is already provided. More or less of the network stack can be incorporated, and more realistic settings can be placed on the edges.

  A benefit of using NS2 over packages like OpNet is that NS2 provides host level simulation as well as traffic simulation, and the code is completely open source. Thus, it can be used for the complete simulation of a scenario, and adjusted to meet the study's specific needs. JNS is the Java port of NS2. Consequently, it is an ideal candidate for improving the realism of the SOMAS simulation, if it is written in Java.
- *Custom:* Since the basic simulation requirements to develop the SOMAS framework are very simple, it is quickest to create a custom built network simulator. By using an object oriented language to write the network simulator, it is possible to attach an existing, and much more indepth package once the SOMAS

framework is complete. This is the approach that is used, due to the simple simulation requirements.

2.7.2 Misc.

- *JGraphT [85]:* JGraphT provides a standard set of graph algorithms. It is useful for manipulating the network graph in the custom simulation.

2.7.3 Multi-Agent System.

- *Swarmfare [100]:* Swarmfare is the in house framework for developing swarms with emergent behavior. Currently, much of it is hardcoded (simulation parameters are values instead of variables) for UAVs, which are different than software agents in a number of important ways. The main sections that need to be reimplemented are the behavior set, the simulation, and the visualization/GUI.

  The advantage of a custom created framework like Swarmfare is that it can be fine tuned for speed. This is important for the problem domain, since the simulations can become very complex and time consuming when there are a large number of nodes and agents. Unfortunately, the framework is too hardcoded and different from the SOMAS problem domain. It cannot be easily converted over, or at least not more easily than building a new application out of other packages.
- *SWARM [49]:* SWARM is the precursor to MASON. It is written in C++. Consequently, it is likely faster than MASON, but it is more unwieldy due to C++ being a more difficult language to use than Java.
- *MASON [75]:* MASON is an open source Java package for simulating multi-agent systems based on the C++ SWARM package, and is very fast. It can also exploit thread level parallelization, by running the different simulation components within different threads and on different cores. To prevent the loss of data

due to an unforeseen exceptions, it can also log the simulation and resume from the results of a log file.

2.8 Visualization

The topic of visualization covers two main areas of the framework: allowing a human to interact with the online swarm and evaluating the results of producing a swarm offline. There are many overlapping concerns between the two areas, though the online visualization also has a time and resource concern.

There are two aspects to the visualization. One aspect is to directly observe the agents' behavior on the network. The other aspect is to observe different measurements on the agent swarm. Measurements can consist of many dimensions, so some form of dimensional reduction is necessary. Most importantly, the measurement visualization needs to visualize the key metrics of self organization and tangled hierarchy. If the visualization of the metrics is paired with the direct visualization of the swarm behavior on the network, then it is possible to visually confirm whether the metrics are correct. Additionally, the data from the metrics can be superimposed on the direct simulation to bring out non-obvious or invisible details.

2.8.1 Self Organization Visualization. It is important to be able to visualize the self organized aspect of the system in order to allow effective human interaction. The best result is if the self organized aspects of the system can be tied to the objectives, so that a person can also specify new SO patterns to further achieve the objectives. There are not many researchers who have a technique for visualizing self organization. This is because there is no consensus on a quantitative way to identify self organization.

Some authors [117] have developed a a visualization technique based on their self organization metric. The idea is essentially to darken the more influential parts of a system and lighten the less influential. When this is done, the structure of the system's organization is evident. The viewer can visually identify whether the authors' self organization metric corresponds with a system that appears to be organized. One visualization looks at the chronology of a self organized system, showing the impact of each causal state on the future. Another visualization takes a snapshot, showing the influence of each part.

2.8.2 SOMAS Visualisation. The criteria for choosing a visualization package are

- integration with existing languages and packages
- ability to take screenshots and record video
- customizable graphics
- automatic network layout generator
- graphing ability (i.e. bar charts, scatterplots, etc.)
- customizable visualization aspect This item means that the visualization is not restricted to a certain set of data, but can be customized to whatever data is appropriate

- *MASON [75]:* Besides the simulation capabilities, MASON also offers a very general visualization API with many pre-constructed visualization techniques, such as 2D, 3D, networks, etc. The visualization runs in a steppable real time visualizer along with the ability to export the visualization as screenshots or movies. The interface itself is also highly customizable and includes the ability to inspect the simulation through the visualization, for example selecting an agent to examine its variable values.
- *JUNG [92]:* JUNG is a network visualization package. It specializes in network layout algorithms, and includes graph generators and basic graph algorithms. Combined with MASON, JUNG provides very nice looking network layouts. JUNG also gives the user the ability to interact with the graphs and create

their own layouts. With SOMAS, JUNG's interaction capability means users can custom design their own scenarios.

- *Swarmfare [100]:* Swarmfare has a very nice GUI, which includes both an interface for setting the algorithm parameters and an ability to visualize the evolved parameters. However, the GUI is hardcoded for UAV simulation, making it difficult to adapt to the task of SOMAS simulation.
- *OpNet [7]:* While OpNet is one of the standard commercial network simulators, and consequently has excellent visualization capabilities, it does not have the ability to visualize host level behavior, which is necessary to completely visualize the SOMAS simulation.
- *NS2 [6] and JNS [131]:* These network simulators can simulate at both the network and host level, so they also have the ability to visualize the simulations. The simulators are not purpose built for agent simulation, so it requires more overhead to construct the visualization than with MASON.

2.9 Testing

Testing requires the ability to process data into an analyzable format, and the statistical tests necessary to test the hypotheses. While both parametric and non parametric tests are useful, non parametric tests are the most general type, and a testing package should have at least these. According to [83] about 30 experiments are required in order to produce statistically significant results. However, in some cases, running an experiment is too time intensive. In this case, as long as the confidence boundaries of the results for the different algorithms do not overlap, statistically significant results are still produced.

To test comparisons between algorithms, it is important to first determine whether the results follow the normal distribution. If they do, then a p test can be used. If the results do not, then a non parametric test must be used. In this project, since the distribution is not known, non parametric tests are used. The tests are the Fisher exact test and the Mann-Whitney test (also known as the Wilcoxon rank-sum test). Both tests assume the runs are independent.

Besides the statistical tests, the testing framework also needs the ability to compare multi-dimensional solutions. This is either accomplished through turning the vector of objectives into a scalar using weights or some other metric, with augmentation, or by comparing the vectors directly. It is difficult to scalarize the objectives since the weights are not always known beforehand. Comparing the vectors directly is a more flexible technique.

The direct comparison usually takes the form of establishing the Pareto front of the solution set, and then comparing a metric based on the Pareto front. Since fronts do not usually have the same number of points, it isn't possible to use a standard dominance ranking. The metric used should also be Pareto compliant, meaning it does not order a dominant front as worse than a non-dominant front [36].

Ideally, the tests should achieve the true Pareto front, PF_{True}. However, this requires a great deal of computational power, and thus only an approximation is likely to be achieved. Consequently, it is also important to be able to determine general characteristics of PF_{True}, so as to understand its general shape, how difficult it is to find the different sections, and the overall sparsity. These characteristics help the decision maker know the constraints on his or her decision making in this problem domain. However, the search landscape to objective landscape mapping function does not necessarily have an inverse function. Thus, a pattern on the objective landscape does not imply a pattern on the search landscape.

- *PISA [24]:* Zitzler's PISA framework [24] is a comprehensive multi-objective development and testing framework. The testing framework consists of a variator and a selector. The variator contains the fitness function, the solution representation, and the solution variation operators. The selector contains the population ranking and selection operators.

Along with these two parts there is also a number of MOEA and statistical metrics that are used to test the hypothesis that one algorithm is more effective than another for a given problem domain. It is simple to add a number of scenarios, variators, and selectors, and then have them automatically tested once the algorithm is complete.

The framework is very general, thus each variator and selector only needs to abide by a simple state machine protocol and file format in order to work together. This allows the user to employ the many selector algorithms available on the PISA site. Additionally, the source code for elements of the framework is generally avaliable, except for one or two exceptions. So, the user has many MOEA templates available to quickly bootstrap their own algorithms without concerning themselves with implementing the required interfaces.

The tests make use of standard *Pareto compliant metrics* to compare Pareto fronts. This means that while the framework is originally meant to support the comparison of different selectors, it can be changed to compare variators, which includes the comparison of evaluation functions. This change consists of switching the position of the variators and selectors in the command line arguments for the run script.

The indicator suite also contains estimated *attainment functions*, which, besides testing, are useful for visualizing where the EA tends to search in the solution space. However, they only work with 2 objective spaces. Even if the objective space for a chromosome contains more than 2 objectives, a pairwise selection of objectives can be used to reduce the objective space for attainment visualization. The objective space can also be reduced by finding linear relationships between objectives, and thus substituting a single objective for related objectives.

2.9.1 MOEA Metrics. In order to determine the comparative effectiveness of different swarms in a scenario, their respective PF_{known} must be compared. The metrics for this comparison should be Pareto compliant [36], in order to have a fair comparison of effectiveness.

Pareto compliance means that the metric, $I : \Omega \rightarrow \mathbb{R}$, is an order preserving function from $(\Omega, \preceq)$ to $(\mathbb{R}, \geq)$, as detailed in formula (19). If the metric is not Pareto compliant, then it may rank a dominated front higher than the dominating front.

$$\forall A, B \in \Omega : A \preceq B \Rightarrow I(A) \geq I(B) \tag{19}$$

$\boxed{!}$ It is important to keep in mind that these metrics are different than the scenario metrics. The scenario metrics measure the online search algorithm effectiveness and the Pareto compliant metrics measure the offline search algorithm effectiveness.

As an example, the following list shows a range of Pareto compliant metrics from [36].

- **error ratio**
- **hyperarea or hypervolume ratio**
- **epsilon (ϵ) metric**
- $R_{1,2,3}$ **metrics**

The PISA framework contains three Pareto compliant metrics: the R_2, ϵ, and hypervolume metrics. These metrics are used in the PISA framework because while each is Pareto compliant, they also measure the dominance of fronts in a different way. The ϵ and R_2 metrics are good for comparing relative dominance of fronts, where the ϵ is more affected by outliers and the R metrics less so. The hypervolume ratio gives an absolute measure related to the theoretical optimal, the objective space origin (for many problems this is an unobtainable goal, thus it is not necessarily the real optimal). These characteristics are discussed in more detail in the following sections.

Epsilon Metric: The Epsilon (ϵ) metric is the minimum $\epsilon \in \mathbb{R}$ value such that any solution $b \in B$ is ϵ-dominated by some solution $a \in A$. This is detailed by

Equation (20) [36].

$$I_\epsilon(A, B) = \min \epsilon \in \mathbb{R} | \forall b \in B \exists a \in A : a >_\epsilon b \tag{20}$$

Where $a >_\epsilon b$ means that a ϵ-dominates b.

The ϵ metric is good for measuring the relative difference between two attainment sets, since it does not require the use of any reference points not in the sets, i.e. the sets are measured against each other directly. However, the ϵ-metric is significantly affected by outliers.

For example, the majority of the vectors in an attainment set, A, may dominate the vectors of attainment set, B. B also contains one or two vectors that significantly dominate vectors in B. Thus, B needs a smaller ϵ value to dominate A, even though A mostly dominates B.

R_2 metric: The R_2 metric is the average difference between the utility values of a reference set and the approximation set, such that utility is measured by a utility function and a set of vectors used for scaling. See Equation (21) [36].

$$I_{R2} = \frac{\sum_{\tilde{\lambda} \in A} u(\tilde{\lambda}, B) - u(\tilde{\lambda}, A)}{|\tilde{\lambda}|} \tag{21}$$

Where $u(A, \tilde{\lambda})$ is the utility of approximation set A, on a scalarized vector, $\tilde{\lambda}$ (a reference point).

The utility is measured by the minimum distance of a point in the set, A, from the reference point. With a large number of reference points, the R_2 metric is a good metric for determining the general attainment of an attainment set. Unlike the ϵ-metric, it is not significantly affected by outliers.

By using both metrics, it is possible to identify greater exploration, such that an attainment has a few remarkably good solutions compared to another attainment. Greater exploration is indicated when the R_2 and ϵ metrics give contrary rankings for two attainment sets. If the difference is big enough, then the data may also contain anomalies.

Hypervolume Metric: The Hypervolume metric is the volume of the minimum values for each objective in the approximation set. This is detailed in Equation (22) [36].

$$\mathbf{ME} \triangleq \left\{ \bigcup_i vol_i | vec_i \in PF_{known} \right\} \tag{22}$$

This metric tells the user how close an attainment set is to the theoretic optimal. In this sense, it is more objective than the previous two metrics. Although, the hypervolume says little about the optimality of the individual points compared to the previous two indicators, since it only can compare the best objectives out of the complete set of points.

2.9.2 Statistical Analysis. The Fisher sign test is not accurate for more than 2 samples, while the Mann-Whitney test is. Since the experiments for this research usually have more than 2 samples, the Fisher sign test is not accurate in such cases. However, it is still used for exploratory purposes. This is because the run count for both statistical tests, which is five runs per scenario, is not sufficient to obtain accurate results, which requires 20 runs per scenario. The smaller run count is used because of time constraints.

Fisher sign test [133]: The Fisher sign test calculates the p-value that two distributions are different based solely on the "signs of the differences between paired observations, not on their sizes." The sample sizes of this test must be the same size. It is accurate for a comparison between two samples, but is inaccurate for more than two samples.

The inaccuracy results from the fact that multiple statistical tests are being run on the same samples, thus the tests are not independent. In such a case, it is necessary to give an idea of how likely at least one null hypothesis is incorrectly rejected, i.e. a

Type I error. In the case of tests that are not independent, the lower bound of this error is given by a' in Equation (23) [83].

$$a' = 1 - (1 - a)^n \tag{23}$$

Where $n = \begin{pmatrix} N \\ 2 \end{pmatrix}$, and N is the number of samples.

For example, if $N = 3$ and $a = 0.05$, then $a' = 0.14$. So, there is at least a 0.14 chance that at least one null hypothesis is incorrectly rejected.

Mann-Whitney (Wilcoxon rank-sum) test [83]: The Mann-Whitney test compares the deviation between the medians of two samples, which may have different sizes. The statistic is calculated by ranking the members of the samples against each other, and adding up the ranks of the smaller sample to produce W_m. The p-value is derived from the range that W_m falls within. The benefit of the Mann-Whitney test is that it can accurately be used on more than two samples, in which case it is more commonly known as the Kruskal-Wallis test.

2.9.3 Visual Data Processing. The visual data processing requirements are fairly simple. First, since each scenario outputs more than 3 objectives, plotting the known Pareto front, PF_{known}, requires dimensional reduction, which can simple be selecting only 3 objectives at a time to plot. Second, the plots need to be capable of incorporating LaTeX markup and be publication quality. There are many very powerful data process packages in existence, but these simple requirements only require the use of Bash and GnuPlot. Besides the data processing for the experiment results, the SOMAS framework requires a clustering algorithm and basic statistical functions. The Weka library provides the clustering algorithm and the Colt library provides the statistical functions.

- *Bash:* Bash is a useful all around scripting language. It is one of the standard scripting languages used in the Linux OS. With the UNIX philosophy of tool chains Bash is an effective glue language. Since the SOMAS framework is composed of a number of disparate packages, glue code is necessary, and Bash is ideal for this task.

- *GnuPlot [135]:* GnuPlot is an open source plotting tool. It is a flexible graphing program that can create publication quality graphs. Its graphs can also be integrated with LaTeX, allowing the use of LaTeX directives to create mathematical formulae and nicely formatted text in the graph.
- *Weka [46]:* Weka provides a very wide range of machine learing algorithms and techniques, along with an integrated development environment (IDE) and grid computing capabilities.
- *Colt [57]:* Colt is a high performance math library that is used by the European Organization for Nuclear Research (CERN). It provides many of standard mathematical and statistical functions necessary for scientific work. Even though there is significant overlap between the Colt library and the Java math library, the Colt functions have all been heavily optimized.
- *R [60]:* R is a standard statistical analysis language. Unlike related packages such as MATLAB and Octave, its language is both object oriented and based on functional languages such as Scheme, making it quite flexible and powerful. Additionally, its speed is comparable to that of MATLAB. Like MATLAB, it has a very large range of libraries, though they are focussed more on statistical operations than matrix operations. Since it is open source, it is also possible to add new functionality to R if needed. Unfortunately, there is not as much documentation for R as there is for MATLAB.
- *MATLAB [78]:* MATLAB is one of the most popular data analysis applications. It is especially good for analyzing matrix based data, and its code is dramatically sped up when properly vectorized. It also has a large number of commercially developed libraries covering a wide range of applications as well as extensive and well written documentation. Unfortunately, MATLAB is a commercial program,

Table 4: Language hierarchy

Level	Examples	Properties	Use case
0	machine code, byte code	direct hardware manipulation, precision, unreadable	writing drivers
1	assembly	precision, readable	speed critical
2	c FORTRAN	structured, imperative, procedural programming, manual memory management	speed essential, legacy
3	lisp, c++, java, prolog	object oriented, functional, declarative programming; static and dynamic aspects; garbage collection	speed important, large scale, framework design
4	python, ruby, perl, javascript	hybrid, completely dynamic	prototyping, glue code, rapid development, throw away code
5	bash, cold fusion, HTML excel, Office VBA, mysql	application tie in, markup, DSL, graphic based	glue code, non-programmer demographic, inter-disciplinary

which can make it difficult to interoperate with non MATLAB based programs. Additionally, the MATLAB programming language is not a suitable language for complex programming task, and neither is it object oriented, making it unsuitable for large scale systems and complex tasks.

- *Octave [43]:* Octave is an open source equivalent to MATLAB. It does not have the GUI functionality of MATLAB, but Octave does provide a close approximation to MATLAB's numerical capabilities. Most MATLAB code can run as is in Octave, with only a few modifications.
- *Sage [123]:* Sage integrates a large number of open source numerical packages, such as R and Octave. It integrates all the packages through the use of Python, a high level programming language.

2.10 Languages

2.10.1 Simulation and Framework Development. The development language, or languages, for the SOMAS framework must meet a number of criteria.

- speed
- libraries
- documentation (if unfamiliar)
- hardware and software compatibility
- rapid development
- extensible code structures (i.e. functional, OO)
- interoperability

Table 4 shows the wide range of languages currently available. The framework should be fairly OS agnostic, but the top level environment is either Windows or Linux, since it adds unnecessary complexity have a hybrid system. Since each chromosome is evaluated by running a simulation, the simulations must run very quickly to achieve useful results in a realistic time frame. High level languages, such as Python and Perl, are not suited for developing simulations because even their compiled code is much slower than lower level languages like C and Java. The language also must be object oriented to both ease development and allow extensive improvement.

The SOMAS framework should be easy to extend and deconstruct. Monolithic frameworks quickly become unmaintainable and unuseful. So, the language must lend itself to loosely coupled code development. Finally, it is beneficial if the language can be used for the whole framework, but this is not a necessary condition. The loosely coupled approach allows a number of different languages to be used as long as they share common interfaces. Thus, the object oriented and speed requirements restrict the language choices to Java and C++.

2.10.2 Scripting. There are a number of scripting languages that have large communities of use. Some are directly tied to an OS, such as the DOS and Bash shells. Others are executed by running an interpreter, for example Perl and Ruby. Since the scripting language needs to easily manipulate files and file structures, as well as process the contents of files using a variety of programs. Consequently, a shell is most appropriate. Thus, the choice of scripting languages comes down to either a Windows based shell such as DOS or PowerShell, or a Unix based shell such as Bash or Ksh.

2.11 Summary

This chapter covers the necessary theory and packages to implement SOMAS. It gives an indepth look at the range of issues involved with each subject. The topic of network security and its great complexity is explained, with the suggestion that it be decomposed using agent oriented design to produce multi-agent based network defense. Models for formalizing multi-agent systems are described.

The difficulty with multi-agent systems is control and the necessary communication for control. If multi-agent systems are controlled with hierarchical communication then they do not scale well. Nature provides many examples of multi-agent systems that scale very well without hierarchical control and communication, while still effectively accomplishing global objectives. The techniques used to generate these multi-agent systems are self organization and entangled hierarchies. Self organization provides the control and communication takes place through entangled hierarchies. To produce self organizing multi-agent swarms, search algorithms are necessary.

Stochastic, global search algorithms are the most general search algorithm. If needed, they can be augmented with deterministic local search algorithms to improve their effectiveness and efficiency. Along with the theory behind the search algorithms, a number of algorithm packages are discussed. To evaluate the multi-agent swarms that are produced by the search algorithm it is necessary to either simulate or emulate the swarm's network behavior.

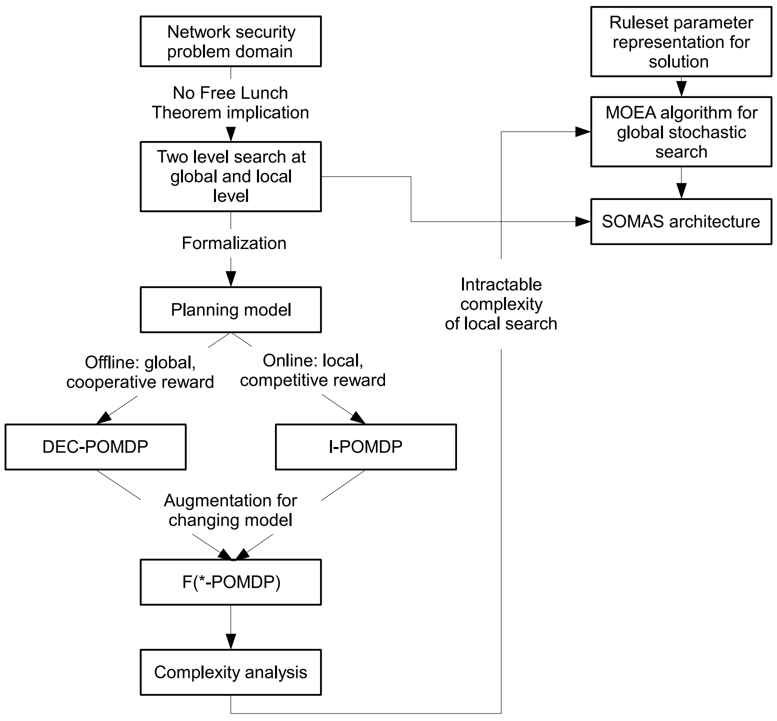
Both network and agent simulation packages are covered. Additionally, visualization and testing packages are listed for analyzing the search results. The languages for programming and scripting the entire SOMAS system are covered, along with criteria for picking the appropriate language. The theory is used to create an architecture design in the next chapter. The packages are used in the following chapter to explain how the SOMAS architecture is implemented.

III. Problem Domain Considerations for Architectural Design

3.1 Overview

The material covered in Chapter II lays the foundation for the network security problem domain analysis and architectural design in this chapter. The design is the basis for "*a mobile, multi-agent system that uses self organization to create entangled hierarchies...for accomplishing network based objectives*" (see Section 1.2).

Figure 2: SOMAS architecture development process


Network security problem domain
No Free Lunch Theorem implication
Two level search at global and local level
Formalization
Planning model
Offline: global, cooperative reward
Online: local, competitive reward
DEC-POMDP
I-POMDP
Augmentation for changing model
F(*-POMDP)
Complexity analysis
Intractable complexity of local search
Ruleset parameter representation for solution
MOEA algorithm for global stochastic search
SOMAS architecture

The overall structure of the chapter is displayed in Figure 2. The general network security problem domain is discussed in Section 3.2, where the structure of the domain and the reward function are described. This structure necessitates a decoupling between global and local reward, and thus Section 3.3 explains the two levels to the problem domain.

Once the problem domain is informally described, it is formally described as a planning problem in Section 3.4, where it is further mapped to POMDP models. The

POMDP models allow the problem domain complexity to be determined, and due to its complexity a global search algorithm is necessary.

The solution representation is discussed in Section 3.5 and the selection of a multi-objective evolutionary algorithm for the global search is discussed in Section 3.6.

Finally, the elements of the SOMAS architecture are covered in Section 3.7.

3.2 Network Security Problem Domain

Table 5: Problem domain elements

SOMAS Agent	Scenario Agent
$< Sensors_{Regional}, Rules, Acutators_{Regional} >$	$< Sensors_{Global}, Rules, Acutators_{Regional} >$
Sensors Real value vectors in limited range $\{(0 \dots 1), \dots, (0 \dots 1)\}$	*Sensors* Depend on scenario
Rules Modus Ponens $condition(sensors) \rightarrow actuator(region)$	*Rules* Modus ponens
Actuator Change agent's region $actuator(region) \rightarrow region' : \Box(region' = region)$	*Actuators* Change agent's region

Network Environment
$< Locations, Edges, Scenario >$ The specifics of each part of the network environment depends on the specific scenario, as in the case of the scenario agents.
Locations $< Agents, Sensors_{Global}, Rules, Actuators_{Local} >$
Edges $< Sensors_{Global}, Rules, Actuators_{Local} >$
Scenario $< Sensors_{Global}, Rules, Actuators_{Global} >$ Can observe and change any aspect of the domain model

Table 6: Problem domain

	Element	States	Action	Transition Functions
1	Agent (A)	$A_S \equiv \mathbf{A_P}$	$\mathbf{A_A}$	$\mathbf{R_S} \times \mathbf{A_A} \times \mathbf{R_S}$
2	Region (R)	$R_S \equiv \{V' : \text{hops}(V, V') <= 1\} \cup \{L : \text{connects}(L, V, V')\}$ $\forall v', l \in R_S : v', l \in N_S$	N/A	N/A
3	Container (C)	$C_S \equiv \{\mathbf{C_P} \cup \mathbf{A}\}$	$\mathbf{C_A}$	$\mathbf{C_S} \times \mathbf{C_A} \times \mathbf{C_S}$
4	Link (L)	$L_S \equiv \mathbf{L_P}$	$\mathbf{L_A}$	$\mathbf{L_S} \times \mathbf{L_A} \times \mathbf{L_S}$
5	Network (N)	$N_S \equiv \{\mathbf{N_P} \cup \{L : \text{connects}(L, V, V') \wedge \text{active}(V) \wedge \text{active}(V')\} \cup \{V : \text{active}(V)\} \cup \mathbb{A}\}$	$\mathbf{N_A}$	$\mathbf{N_S} \times \mathbf{N_A} \times \mathbf{N_S}$
6	Scenario (S)	$S_S \equiv \{\mathbf{S_P} \cup N\}$	$\mathbf{S_A}$	$\mathbf{S_S} \times \mathbf{S_A} \times \mathbf{S_S}$

$$\text{Global reward function} : \mathbf{S_S} \times \text{Time} \rightarrow [-\infty \ldots \infty]$$
$$\text{Local reward function} : \mathbf{A_O} \rightarrow [-\infty \ldots \infty]$$

S subscript	A state.
P subscript	A parameter.
A subscript	An action.
O subscript	An observation, which only agents have.

The network of network security is a very general concept. While it can refer to the standard wired network, it can also refer to wireless ad hoc networks as well as logical networks that are instantiated in software. As such, the problem domain is defined symbolically in such a way that it can be mapped to any such network. It is important to define the problem domain symbolically in this way in order to determine different general problem characteristics, such as runtime complexity, see Section 3.4.4, or scenario dynamics, see Section 5.6.

To symbolically defined the problem domain, first the elements and their relations are identified in this section. Then, in Section 3.4, the elements are mapped to formal models in order to produce complexity characteristics. The problem domain elements are covered in Table 5.

The problem domain is composed of two primary elements: agent (A) and environment ($\mathcal{E}$), which in the case of this research is a computer network (N). The objective in this problem domain is, in general, to maximize the control of the network's owner. This encompasses many different network metrics, ranging from the high level metrics such as mission accomplishment, mid level metrics such as quality of service, and low level metrics such as intrusion detection. For a fuller discussion, see Section 2.2.5.

The important thing to note is that all metrics are subject to a human agent's will, and the agent's goal may not be something that can be measured in regards to the network. Thus, it may be mathematically impossible to represent to highest network objective in terms of a quantifiable metric. Ultimately, it comes down to human judgement. Consequently, solutions to the network security problem should leave as much flexibility to human judgement as possible, as the SOMAS approach does by allowing any goal to be defined for SOMAS that can be qualified in terms of the elements in table 5. Thus, SOMAS is a form of intelligence augmentation, as discussed in Section 2.5.8.

For the purposes of this research all hardware in the network is assumed to only be controlled by software. Physically securing the network hardware is outside the scope of a computer science solution. Additionally, the container model assumption implies that only relevant software state information is accessible for reading and writing. If all locations in the network are constantly connected, then the container can be considered to encompass the whole network. However, in general, this is an invalid assumption. Thus, containers are segmented according to the links connecting hosts in a network. Containers are described in further detail in Section 2.3.5.

The network graph is composed of vertices (V) and links/edges (L). The vertices are equivalent to agent containers (C) for the purposes of this research. Table 6 further symbolically details each problem domain element, and the following list explains each symbol. Each state element is indexed with a time-stamp. Thus, the scenario state

$S_S(t)$ consists of all scenario state elements with time-stamp t. To reduce notational confusion, the (t) index is only used on the global scenario state S_S.

1. An agent's state consists of a set of parameters. Its actuators can affect its region.
2. A region state is defined as the agent's local container state, the state of neighboring containers that are one hop away, and the connecting links. All containers and links in the region state must also be in the network state.
3. A container's state consists of a set of parameters and all the agents in the container. Its actuators can affect only its own state.
4. A link's state only consist of a set of parameters, and its actuators can only affect its state.
5. A network state contains a set of parameters, all active nodes, and their connecting links. Network actuators can affect the whole network state.
6. The scenario state contains scenario parameters and the network state. The scenario actuators can affect the whole scenario.

The objective for the scenario is described by the reward functions. The reward can either be specified at the global or local level. The objective can also be considered the swarm's behavior, since the swarm's is meant to accomplish the objective. However, behavior can also refer to the swarm's ruleset, in which case a swarm's behavior is not necessarily the same as the objective since the ruleset may not achieve the objective. There are many objectives that the swarm can achieve, and a list is proposed in Appendix F. The specific behaviors that are examined in this research are:

- Unobtrusive network activity
- Self organization
- DDoS prevention

- Avoiding occupied nodes
- Removing intruders from the network
- Competition between friendly and malicious agents
- Protecting the network information flows

The following is a simple example showing how the problem domain symbols are used to define a behavior. Each objective is meant to be minimized. The behaviors are further described and formalized in Section 5.5.

Vital Vertex Identification: The behavior is to identify the vital vertices in a network. The vital vertices are the minimal set of vertices in a network that cause the greatest network degradation when they are removed. The extent of network degradation is measured by comparing the cumulative length of the shortest path for all pairs of vertices and the number of disconnected vertices before and after the vertices are removed.

This is almost the same as the metric of network degradation in [19]. The difference lies in counting the number of disconnected vertices. The original metric did not do this, and only has a single objective. Using a single objective raises the problem of rating the cost of disconnected vertices. However, making the metric produce a double objective measure eliminates the ambiguity.

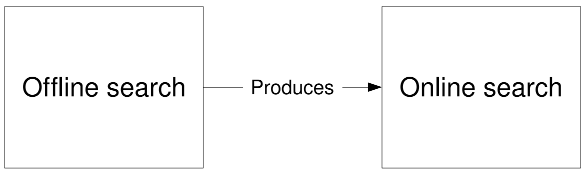
Finding the vital vertices in a network is an NP-Hard problem [19]. The difficulty of the problem is increased by the fact that the problem information is only partially observed, each agent only has a partial view of the whole network. Equations (56-57) measure the network degradation. Equation (58) measures how concisely the agents can identify the vital vertices.

$$\sum \forall u \forall v (shortestPath(u,v) \leftrightarrow u,v \in \mathbf{C} \wedge u \neq v) \tag{24}$$

$$\sum \forall u \forall v (pathExists(u,v) \leftrightarrow u,v \in \mathbf{C} \wedge u \neq v) \tag{25}$$

$$|\mathbf{ID}| : \mathbf{ID} \in \mathbf{C} \tag{26}$$

Figure 3: Offline to online search



Where **C** is the graph's vertex set, which is equivalent to the graph's container set. **ID** is the set of vertices identified as being vital vertices.

3.3 Offline and Online Swarm Generation

Since most real networks are subject to extensive and continuous change, especially when its security is being compromised through intrusion or attack, a swarm that maintains a fixed ruleset will perform badly in the general case. To adapt to a changing environment, a swarm needs to search for the best ruleset in a given environment state. One way to do this is with reinforcement learning [88], which learns a utility function. However, this assumes local access to a global reward, which may not be possible.

According to the No Free Lunch Theorem [138], any two search algorithms are going to be equivalent to each other across all possible fitness landscapes, and thus no better than Monte Carlo sampling. This means learning work needs to be done in order to guarantee the online search algorithm has better than random behavior in the general case. Consequently, if there is an offline stochastic search at the global level for the online search algorithm, *then it is possible to mathematically guarantee an online search algorithm that is better than or equivalent to all other online search algorithms, in a given range, for all fitness landscapes.*

As a result, there are two stages to the swarm's lifecycle on the network: when it is generated and when it is executed. These stages are shown by the two panes in Figure 3. There are two problem representation (model) domains and two search (algorithm) domains, respectively corresponding to the offline generation and online runtime.

As described in Section 2.4.1, the agent schema in [88] is equivalent to the POMDP model. The DEC-POMDP and I-POMDP models are the most comprehensive for characterizing multi-agent systems. The differing perspectives match the two stages in the swarm's lifecycle, thus they are chosen as being the most suitable for deriving theoretical characteristics for the problem domain, such as problem complexity. Due to the nature of the reward function in either case, a DEC-POMDP model is used for the *offline problem representation* where the agents can be modelled as cooperative and an I-POMDP model is used for the *online problem representation* where agents may be competitive. Both are additionally augmented to be F(*-POMDP) models in the case where agents can be added and removed from the environment, or the environment can change.

The reason why the agents in the online stage may be competitive is because an agent only has direct access to its own observations. To access the observations of other agents requires communication, and communication affects the global state, which can affect the agents' local rewards. Since the global process in a computation environment can be Turing complete, an agent may be incapable of calculating the global state from its observations. Therefore, is not always possible in the general case for the agent to know how its local observations map to the current local rewards of other agents. Consequently, agents may minimize the local rewards of each other through maximizing their own local rewards, and thus compete.

As mentioned in Section 2.4 it is intractable to generate optimal or approximate policies from these POMDP models in all but the simplest cases. Thus, since a POMDP policy is a ruleset, rulesets are not restricted to POMDP policies, the solution domain in section 3.5 is generalized to a ruleset.

3.4 Mapping Problem Domain to POMDP Models

The problem domain is a planning problem [88]. All scenario state transitions only depend on the previous state, so the Markov assumption holds for this problem domain. Thus, transition functions only require the previous state, as shown in the *Transition Function* column of Table 6. As discussed in the previous section, there are two perspectives the multi-agent system can be viewed from: the global or local perspective. From the global perspective, the viewer has access to the global state and corresponding reward. However, from the local perspective, the viewer only has access to the local state. Since the local state does not necessarily embed any global information, it does not necessarily provide any global reward information. Thus, from the local perspective agents may have to compete with each other to maximize the swarm's global reward. Since the two views imply two different kinds of reward functions, two different representational models are necessary: the DEC-POMDP and I-POMDP models.

3.4.1 DEC-POMDP Model. [22]

The DEC-POMDP model is a POMDP model for multiple agents that can interact with the same environment and share the same global reward function. This model represents a global perspective of the multi agent system, since only from the global perspective can the group reward be known. The use of the model here does not exactly match that of [22] since only the elements necessary to derive the problem domain complexity are used.

$$\Psi_{DEC} :< S, \mathbf{A}, T, \mathbf{\Omega}, O, R > \tag{27}$$

- S consists of all scenario states $P_S \in P$. This includes the state of each vertex (V) and link (L), extra scenario info, etc.
- $\mathbf{A}$ is a set of sets, where each set is drawn from $\mathbb{N}_A$, the universe of possible agent actions including the null action. Each set consists of possible aggregate actions of agents in the swarm. It is defined according to formula (28), which shows every action set is the same size and every agent accomplishes exactly one action. a_i is the action of agent i. n is the number of agents in the swarm.

$\{\beta\}^\alpha$ means the closure of alphabet β, where all strings in β are of length α.

$$\forall A[A \in \mathbf{A} : A \in \{\mathbb{N}_A\}^n \wedge \forall a_i, a_j \in A\{i \neq j\}] \tag{28}$$

Each action in $\mathbb{N}_A$ is composed of agent actuators of the form $\delta(\Phi_S, \gamma)$ where δ is a function with possibly irreversible side effects, Φ_S is a set of elements $\forall \phi_S\{\phi_S \in \Phi_S : \phi_S \in p_S\}$, and γ is a set of parameters. The scenario can have rules as well, but these are implied in the state transition and do not need to be explicit in $\mathbb{N}_A$.

- T maps state/action set pairs to states, according to formula (29).

$$T : S \times \mathbf{A} \rightarrow S \tag{29}$$

- $\mathbf{\Omega}$ is a set of observation sets, defined similarly to $\mathbf{A}$, detailed in formula (30).

$$\forall \Omega[\Omega \in \mathbf{\Omega} : \Omega \in \{\mathbb{N}_\Omega\}^n \wedge \forall \omega_i, \omega_j \in \Omega\{i \neq j\}] \tag{30}$$

- O maps T element/observation set pairs to probabilities, according to formula (31). An element f_e of function f is pair $(x, y) : f(x) \rightarrow y$.

$$O : T_E \times \mathbf{\Omega} \rightarrow (0 \ldots 1) \tag{31}$$

- R is the global reward, identical for every agent since it is global, accruing a value in $\mathbb{R}$ for each element of T, detailed in formula (32).

$$R : T_E \rightarrow [-\infty \ldots \infty] \tag{32}$$

As an example of how this model can be used, consider the scenario where SO-MAS agents have to identify and eliminate malicious agents from a network. The environment states, observations, and actions are straightforward. A state transition in the T function is, for example, a SOMAS agent changing its location. An observation in the O function is when the SOMAS agent at a location runs an intrusion sensor on the location. The state (there is a malicious agent at the location) and the scan action have a probability that the SOMAS agent will actually identify the presence of the malicious agent. Finally, an example of an element of the R function is when the malicious agent has been located, the SOMAS agent successfully deletes the malicious agent.

3.4.2 I-POMDP Model . [40] Unlike in the case of SOMAS generation with global information, SOMAS generation with local information cannot make use of a group reward, since the agents in the swarm cannot necessarily have a global view of the swarm and environment state. Consequently, each agent has a local reward. Additionally, since observations are limited and local competition can maximize the global reward, agents cannot completely trust each other. Thus, they need to be able to examine each other, have beliefs about each other, and have local reward functions. These capabilities are key to the I-POMDP model. The use of the model here does not exactly match that of [40] since only the elements necessary to derive the problem domain complexity are used.

$$\Psi_I :< IS_i, A, T_i, \Omega_i, O_i, R_i > \tag{33}$$

- IS_i pairs environment states with agent tuples, where, as opposed to the DEC-POMDP model, $\{S' : A\backslash\emptyset, S\}$, there is no agent element in the state tuple. *The notation* $\{\alpha\backslash\beta, \Gamma\}$ *in the previous formula means to replace literal* α *with literal* β *in string* Γ. The set of agent tuples includes all agents except for agent i. Thus, a pair in IS_i follows the schema $(s', \psi_\mathbf{I})$, where $\psi_\mathbf{I}$ is a set of instances of agent schemas with $|\psi_\mathbf{I}| = n - 1$ and n is the number of unique agents in Ψ_I.
- A is a non empty set in $\mathbb{N}_A$, $\{A : A \neq \emptyset \wedge A \in \mathbb{N}_A\}$.

- T_i describes the standard POMDP transition function for agent i, as detailed in formula (34).

$$T_i : IS_i \times A \to IS_i \tag{34}$$

- Ω_i is agent i's observation of its environment, which technically can include direct observation of other agents' states, but is usually not allowed. However, this is quite possible at least amongst SOMAS agents that share the same container, so an observation can indeed include the direct state of other agents.
- O_i, similar to the DEC-POMDP, maps pairs of T_i elements and observations to probabilities, as detailed in formula (35).

$$O : T_{i_E} \times \mathbf{\Omega} \to (0 \ldots 1) \tag{35}$$

- R_i, unlike its corollary in the DEC-POMDP, is an agent specific reward, which may or may not cooperate with that of an arbitrary agent j. This is due to the fact that the optimization of the global reward may require competition amongst swarm agents. The function is detailed in formula (36).

$$R_i : T_{i_E} \to [-\infty \ldots \infty] \tag{36}$$

3.4.3 F(-POMDP) Model.* Both the DEC-POMDP and I-POMDP models have an important limitation in that they are static and finite. Thus, while they can represent a decreasing number of agents, they cannot represent a limitless increase in agents. The models also cannot represent agents that can change their own or other agents policies. In order to get past these obstacles, it is necessary to introduce a transition function between POMDP models. This transition function is not found in the literature, but it allows the POMDP models to adapt to changes in environment. The F(*-POMDP) notation is used because it represents that the augmentation is a function mapping POMDP models to POMDP models, which are generally referred to with the asterisk.

For another demonstration of the utility of this POMDP representation in different simulation and testing domain, please see appendix G where the QuERIES model is examined and the applicability of the F(*-POMDP) model is recommended.

T' is the model transition function described in section 2.4.1, which maps model/action set pairs to models. The notation $\{\alpha\backslash\beta, \Gamma\}$ works as described in Section 3.4.2.

$$\{T\backslash T', *POMDP\}|T' : *POMDP \times \mathbf{S} \times \mathbf{A} \rightarrow *POMDP \tag{37}$$

T' is necessary for both the DEC-POMDP and I-POMDP models of SOMAS production since agents are added and removed from the environment in both cases.

3.4.4 Complexity of Deriving Model Policies. From the POMDP model complexity results in section 2.4 it is clear that even though the DEC-POMDP and I-POMDP formalizations capture the SOMAS problem domain, it is impractical at this stage of the cutting edge to generate the SOMAS policies from either the DEC-POMDP or I-POMDP models. Since T' is potentially Turing complete, i.e. finite state machine with 2 pushdown stores, it is susceptible to the halting problem. Therefore, it is logically impossible to solve F(*-POMDP) models in the general case algorithmically. However, this does not rule out the possibility that the models can generally be solved mathematically, i.e. with proofs. If a policy solution is generated from the F(*-POMDP) model, it becomes a part of the model since agents can manipulate the policies of agents, so an infinite horizon policy has to be a fixed point function.

As can be seen in the case of each model, while they succintly capture the theoretical aspects of each stage of SOMAS production, optimal or approximate policies cannot be derived from the models in general in a tractable manner. This means a different representation and policy generation algorithm is required for the solution domain.

3.5 Solution Domain

A solution consists of a behavior that accomplishes a certain objective. The behavior depends on the agent rulesets ($\mathcal{R}$) and the environment. A behavior from the global perspective is a state/action tuple to probability mapping function, formula (38). A behavior from the agent perspective is a rule (r) to probability mapping function, formula (39). An objective is an environment tuple set to reward mapping function, formula (40). $\wp(\Omega)$ is the powerset of Ω.

$$\mathcal{B}_g : S \times A \rightarrow (0 \dots 1) \tag{38}$$

$$\mathcal{B}_l : r \rightarrow (0 \dots 1) \tag{39}$$

$$\mathcal{R} : \wp(S) \rightarrow [-\infty \dots \infty] \tag{40}$$

The intent is to find a solution $\mathbb{S}$ given an initial definition of self organized behavior that maximizes the reward, and this can be done at either the global or local level, (see formula (41)). Thus, a solution consists of a set of rulesets, out of all possible rulesets, that maximizes the reward. $\sum f \circ g$ is the sum of the range of $f \circ g$.

$$\forall b_1 \forall b_2 \{b_1, b_2 \in \mathcal{B}_{[gl]} : \sum b_1 \circ \mathcal{R} \geq \sum b_2 \circ \mathcal{R} \rightarrow b_1 \in \mathbb{S}\} \tag{41}$$

Where a rule is sequence of operators (δ) and operands (γ). Rulesets (R) are defined according to formula (42).

$$\forall r \exists \delta \exists \gamma \{r \in R : \delta \in r \wedge \gamma \in r\} \tag{42}$$

If the rules and the sequence of operators in the rules of a ruleset are fixed, then finding the solution at the local level is detailed in equation (43), which states that a solution is a fixed size set of operands. A fixed ruleset is chosen in order to reduce the state space complexity and based on the assumption that the required rulesets are accessible to the SOMAS creator.

$$\text{argmin}_{\gamma \in b_R}(\sum b_R \circ \mathcal{R}) \tag{43}$$

Where b_R is the behavior formed by ruleset (R). In this case, the argmin function is equivalent to deriving the policy for a POMDP model, since the set of possible actions is fixed.

3.6 Algorithm Domain

To develop a good search algorithm, first it is important to characterize the best type of search algorithm to use. Table 7 shows a way of subdividing search algorithm types so as to make an appropriate algorithm selection based on what is known or not known about the problem domain. When the fitness landscape is not well known an analysis of the problem and solution can give a general rule of thumb for additional fine tuning of algorithm selection.

Once the algorithm is selected, the objective space needs to be defined. Generally, the objective space can either be single or multi-objective. While single objective spaces have been extensively explored, multi-objective problems are less well known in computer science domains, though they are well studied in the field of operations research and other engineering disciplines.

3.6.1 General Algorithm.

Finding optimal policies for either the DEC-POMDP or I-POMDP models is intractable because the problem is not greedy in structure such that an optimal solution can be built from optimal subsolutions [98] and the search landscape is not known. The search landscape is not known since the POMDP model being used is generally unsolvable, see Section 3.4.3. Therefore, a global solution space search is required as shown by table 7.

Algorithm 1 Deterministic global solution search

```
  $\mathcal{S}_i^p \leftarrow \{init()\}$;
  $\mathcal{S}_o^f[0] \leftarrow \{\}$;
  $t := 0$;
Ensure: $\forall s^p\{s^p \in \mathcal{S}[t] \rightarrow s^p \in \mathcal{S}_i^p\}$
Ensure: $\forall s^f\{s^f \in \mathcal{S}^f[t] \rightarrow s^f \in \mathcal{S}[t] \wedge \mathcal{F}(s^f)\}$
Ensure: $\forall s^f\{s^f \in \mathcal{S}_o^f[t] \rightarrow s^f \in \mathcal{S}^f[t] \wedge \mathcal{C}(s^f, \mathcal{S}_o^f[t-1])\}$
  while $\mathcal{O}(\mathcal{S}_o^f[t], t) \neq TRUE$ do
    $t := t + 1$;
    $\mathcal{S}[t] := generate(\mathcal{S}_i^p, \mathcal{S}[0 \ldots t-1])$
    $\mathcal{S}^f[t] := feasible(\mathcal{S}[t])$
    $\mathcal{S}_o^f[t] := select(\mathcal{S}^f[t], \mathcal{S}_o^f[0 \ldots t-1])$
  end while
```

Algorithm 2 Stochastic global solution search

```
  $\mathcal{S}_i^p \leftarrow \{init()\}$;
  $\mathcal{S}_o^f[0] \leftarrow \{\}$;
  $t := 0$;
Ensure: $\forall s^p\{s^p \in \mathcal{S}[t] \rightarrow s^p \in \mathcal{S}_i^p\}$
Ensure: $\forall s^f\{s^f \in \mathcal{S}^f[t] \rightarrow s^f \in \mathcal{S}[t] \wedge \mathcal{F}(s^f)\}$
Ensure: $\forall s^f\{s^f \in \mathcal{S}_o^f[t] \rightarrow s^f \in \mathcal{S}^f[t] \wedge \mathcal{C}(s^f, \mathcal{S}_o^f[t-1])\}$
  while $\mathcal{O}(\mathcal{S}_o^f[t], t) \neq TRUE$ do
    $t := t + 1$;
    $\mathcal{S}[t] := generate(\mathcal{S}_i^p)$
    $\mathcal{S}^f[t] := feasible(\mathcal{S}[t])$
    $\mathcal{S}_o^f[t] := select(\mathcal{S}^f[t])$
  end while
```

Table 7: Search algorithm types

	Known	Unknown
Local	Partial Deterministic (Depth First)	Partial Stochastic (Random Walk)
Global	Full Deterministic (Breadth First)	Full Stochastic (Monte Carlo Sampling)

Algorithm 1 is the general format of global solution space search algorithms. A superscript p means a partial solution and a superscript f means a feasible solution. Feasible solutions encompass complete solutions, since a solution must be complete in order to be feasible. A subscript i signifies the input set and a subscript o signifies the output set. $\mathcal{S}$ is a set of solutions, and these can be either partial or feasible. $\mathcal{O}$ is the objective, which is a function of both the output solution set and the current iteration count. $\mathcal{F}$ is the feasibility function, which ensures the solution is feasible. Finally, $\mathcal{C}$ is a choice function that selects the best solutions to be added to the output set from the current set of feasible solutions.

Both *generate* and *select* are also functions of their respective sets' pasts in order to avoid cycles in the search for solutions. However, if a stochastic search algorithm is used, then cycle detection is unnecessary, and the search algorithm can be simplified To algorithm 2. As long as a stochastic search can always sample the *correct portion of the solution space at the correct time* to fulfill the objective function, then the algorithm is guaranteed to halt [88].

This result holds for all finite search graphs, and infinite planar search graphs. However, if the graph is non planar, then the guarantee no longer holds [58]. While the solution search space is not a planar graph, the network goal states search space can be, if the network graph is finite or planar. *Since all real world networks are finite, then any network goal state search that incorporates a random walk is guaranteed to eventually find the goal state.* However, while halting is guaranteed, efficiency is not.

Table 8: Algorithms mapped to fitness landscape characteristics

	Hilly	Craggy
Smooth	Tabu search	Evolutionary strategies
Rough	Simulated annealing	Evolutionary algorithm

Therefore, it is important to also include memory and/or a heuristic in the search algorithm.

The field of metaheuristics provides numerous general approximation search techniques, such as simulated annealing, Tabu search, evolutionary algorithms, and ant colonies [82]. Evolutionary algorithms (EA) are most suited for the SOMAS problem domain because they search the global solution space and an effective swarm needs to incorporate multiple kinds of behaviors (building blocks).

Table 8 shows the general mapping of global search algorithm types to landscape characteristics. See Section 2.6.1 for a discussion of the different kinds of characteristics. Systems with *positive feedback loops* have the potential to become chaotic and unstable. Feedback is one of the main characteristics of self organization, see Section 2.5.5. Consequently, the SOMAS rulesets can have chaotic effects. This means minor changes in the rule parameters can have great effects, and major changes in the rule parameters can have small effects. So, the slopes to the optima are very rough. As discussed in Section 1.5, *the potential for instability is a serious concern when considering whether to use SOMAS in a given problem domain.*

Depending on the complexity of the scenario and the corresponding goal, the best SOMAS behavior may be composed of many sub-behaviors. Sub-behaviors imply agent rulesets that are significantly different from each other. But sub-behaviors contribute to the reward of the overall behavior to different degrees. This means a complex problem usually has a very mountainous search landscape.

Based on this analysis of the SOMAS search landscape, Table 3 states simulated annealing and evolutionary algorithms are the most suitable global search algorithms for the SOMAS problem domain. Since hilly landscapes are a subgroup of mountainous landscapes, evolutionary algorithms can cover a wider variety of landscapes than simulated annealing. So, an evolutionary algorithm [16] is selected for the search algorithm. As discussed in Subsection 3.6.4 the objective space is multi-objective. So, the evolutionary algorithm must be a multi-objective evolutionary algorithm.

3.6.2 Genotype. Due to the design decision in section 3.5 to fix the operators in the rule set, the solution consists of a vector of operands. Since the number of operands each operator requires is fixed, and the number of operators is fixed, the size of the operand vector is also fixed. The design decision for the search algorithm is to use evolutionary algorithms (EA) for both the off and online SOMAS production.

As discussed in [107], there are numerous EA solution representation techniques. The binary string and real value vector options are the most relevant. Since all representations are encoded as binary strings at the machine code level, the binary string representation is more general and encompasses the real value vector representation. Additionally, since the evolutionary algorithm is considered the most suitable for the fitness Landscape 3.6.1, and traditionally it uses a binary string representation for its solutions, the binary string is a good choice for the solution representation.

There are a number of issues involved with using a binary string representation. First, it is important that the hamming distance between genotypes is related to the distance between phenotypes. This ensures that children are related to their parents. If this were not the case, then the operators would violate the requirement for variation operators in formula (46), since the solution generated by the variation based on a set of parents would be equivalent to the solution generated given an arbitrary sample from the solution space. Such a result would make the search no better than a Monte Carlo sampling of the solution space. The Hamming distance problem can be solved, to a degree, by using a gray coding [15]. Gray coding takes a standard binary encoding of some value and, using a reversible function, transforms it so if any arbitrary bit is changed, the new value only differs from the old value by

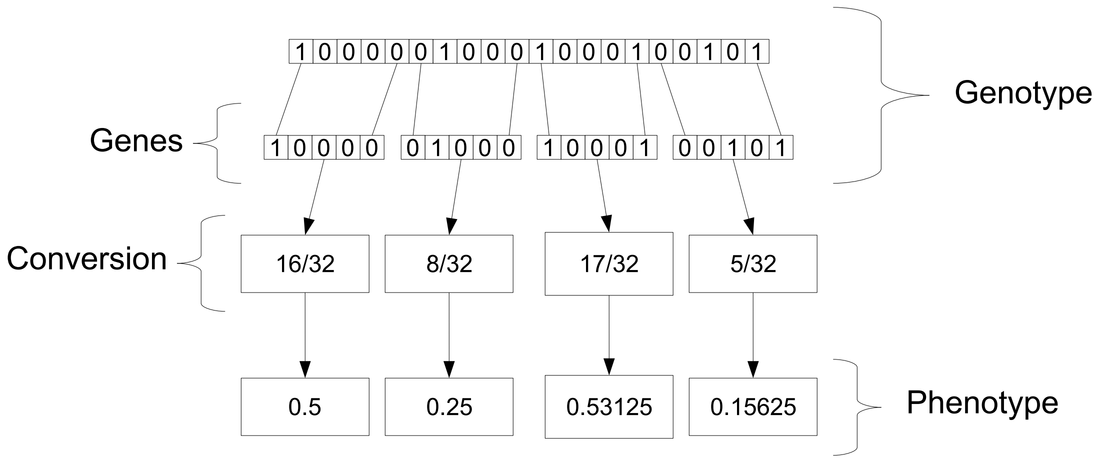
1. It is an $O(n)$ operation, where n is the length of the bit string, so gray coding does not add significant overhead to the algorithm.

Second, the use of genetic operators means the schema theorem is applicable. Since the distance between bits in a binary string schema affects the likelihood the schema can be disrupted by variation, the epsistasis of good blocks is a concern. This problem can be dealt with in a number of ways. At the representation level, a messy representation can be used, where alleles in the chromosome no longer are placed in positions (though they retain positional information), eliminating the problem of bits being inserted or removed from schema. At the operator level, this problem mainly occurs when multipoint crossover is used, since all bits between the crossover points are potentially changed. Consequently, if uniform crossover is used instead of using multipoint crossover, then this is equivalent to using a messy representation, since the position of bits only matters insofar as the position dictates the solution's phenotype.

Often it is the case that the genotype does not map to a feasible phenotype. This is true for the SOMAS problem domain as well. There are a couple of ways of dealing with infeasible solutions. Perhaps the simplest is to merely discard the solution. However, even infeasible solutions can contain good building blocks. Another technique is to penalize the objective function. The problem with this technique is that perhaps the building block is very good. If so, then the building block may still not be selected since its objective value has been decreased. A good solution to these problems is to add feasibility as an objective. Then, if the building block contributes to good objective values, this is fully represented in the objective space. The feasibility metric ensures that even with good building blocks the infeasible chromosome will still be penalized in relation to all feasible chromosomes.

3.6.3 Phenotype. In section 3.5 it is stated that a solution consists of a fixed size set of operands. Since the genotype is of a fixed size, that means the operands are of a finite fidelity. In order to make the chromosome as flexible as possible, it

Figure 4: Genotype to phenotype conversion



is important to make the phenotype values capable of being used in any operand position.

Consequently, the genes all map to a floating point value in the range $(0 \ldots 1)$, as shown in Figure 4. Thus, the operators and operands are loosely coupled and the operands are invariant with regard to changes in the chromosome. For instance, the gene size can be increased and the meaning of the phenotype remains the same tothe operators, only the fidelity is increased. The operators, in turn, can convert the value to the necessary form, such as a negative number or scaled between $(0 \ldots \infty]$ using an asymptotic scaling function.

3.6.4 Objective Space. Once the phenotype is generated, the resultant swarm behavior is evaluated in the simulator. Often, multiple metrics are necessary to characterize a desired behavior. This makes the problem a multi-objective problem (MOP).

There is often a tradeoff between objectives; it is usually impossible to find a single solution that minimizes each objective. Such problems are common in the field of operations research and engineering. In operations research, there are two main techniques for dealing with multi-objective optimization. The objectives are either reduced to a single objective, augmented, or solutions are compared based on their

Pareto dominance [36]. Since the latter technique is more general, it is used to rank the solutions into Pareto equivalent sets, as detailed in equation (44).

$$\mathbf{PF}_{TRUE} = \forall f(\text{argmin}^{P}_{s \in \mathcal{S}}(f(s)) : f \in \mathcal{F}) \tag{44}$$

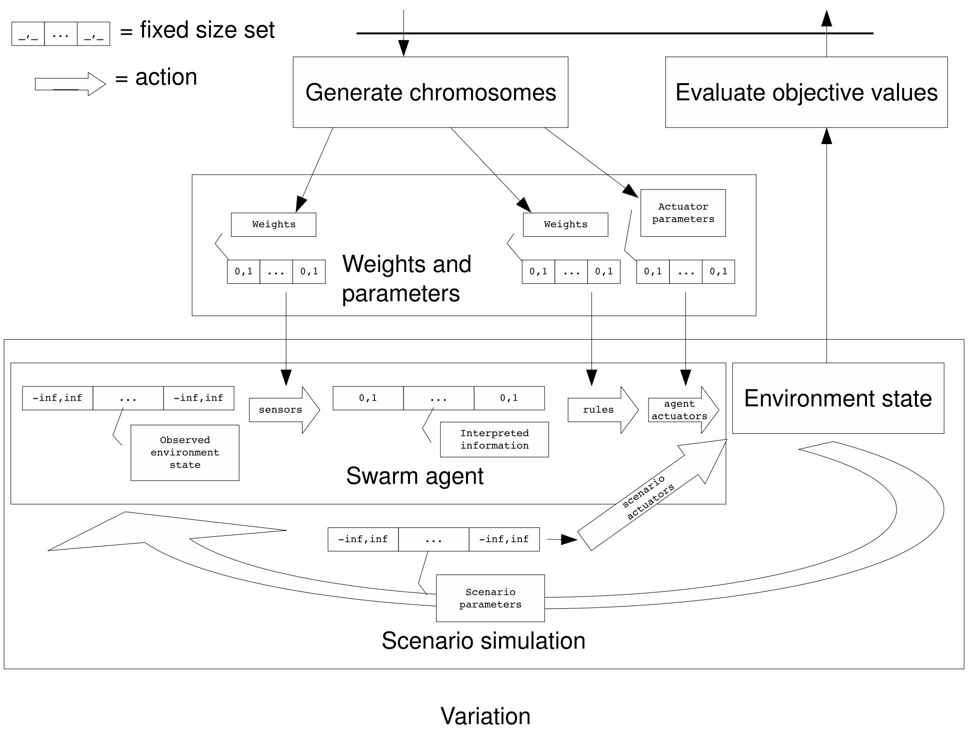
The best set is called the $\mathbf{PF}_{TRUE}$, since it is the Pareto dominant set out of all the solutions, also known as the Pareto front. In the case of a search for an approximate solution the solution set is $\mathbf{PF}_{KNOWN}$ since it is the known best set, but not necessarily the optimal set. argmin^{P} is an argmin function that uses a Pareto dominance ranking metric to find the Pareto dominant set of solutions.

Multiple objectives in MOPs do not generally map in a regular pattern to the decision space. Additionally, the curse of dimensionality implies the objective vector mean lies in a radius around the vector composed of the mean of each objective, and the radius increases with the number of objectives [23]. Therefore, it is useful for a MOEA to be more exploratory as the number of objectives increase. At the same time, as the algorithm nears the PF_{TRUE} it must become more exploitative in order to actually reach the PF_{TRUE}, or get significantly close.

3.7 SOMAS Architecture

3.7.1 SOMAS Algorithm Design. As shown in Figure 6, there are three primary sections to the SOMAS production system, per the standard functions in a evolutionary algorithm [16]. While there are four standard functions in an EA, the figure only contains three since the recombination and mutation functions are grouped into the more general variation function. The evaluator gives each chromosome an objective value set ($O : O \neq \emptyset$). $|O| > 1$ in the case of multi-objective evolution, as is generally true in SOMAS production. The objectives are derived according to an objective view of the global behavior reward, but they can be transformed in order to affect the search. For example, proportional selection [16] scales the objective values so when a plateau is reached selection is still highly elitist.

Figure 5: SOMAS evolution concept

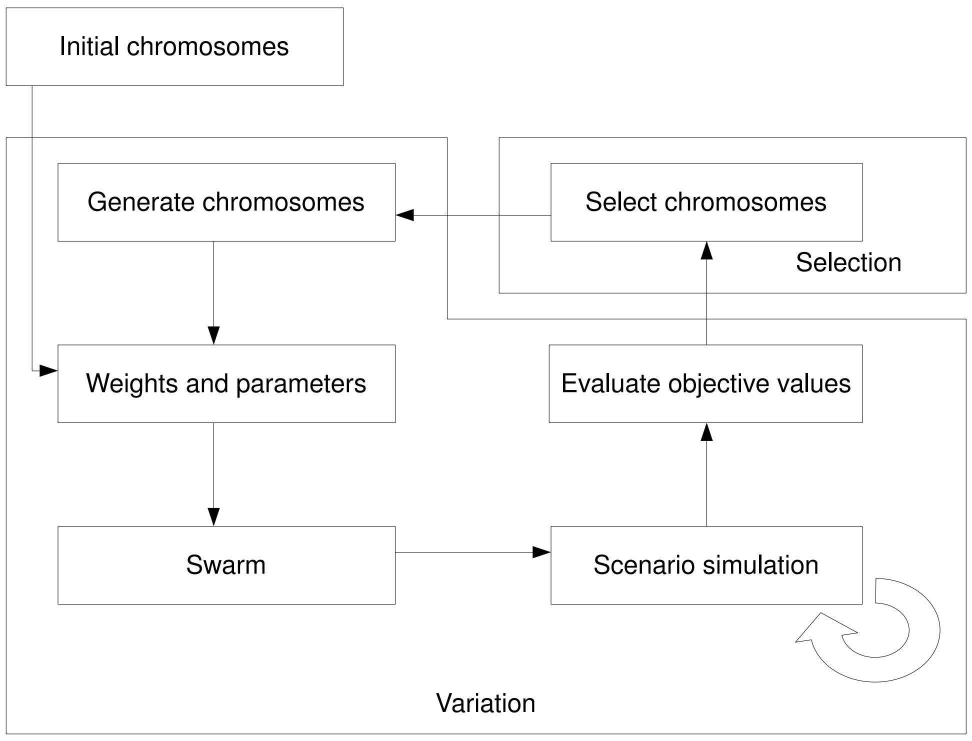


In the case of the SOMAS production system, the objective values are obtained from a set of metrics that measure the simulation. The selector makes use of an ordering metric to select the fittest solutions for use in generating new solutions. Selection is straightforward if only single objective selection is required. Because fitness values are usually numeric, then a simple ordering metric is numerical comparison. If the fitness values are non numeric, then other metrics can be used, such as lexicographic comparison [16].

Comparison is more complex in the case of multi-objective problems, as described in Section 3.6.4. Numerous algorithms have been developed to deal with multi-objective selection. Due to the loose coupling inherent in the framework design, any selection algorithm can be placed in the selection section without affecting the other sections.

The variator takes the solutions selected by the chromosome and uses a set of variation operators on the solutions to generate a new set for evaluation. The

Figure 6: SOMAS MOEA design



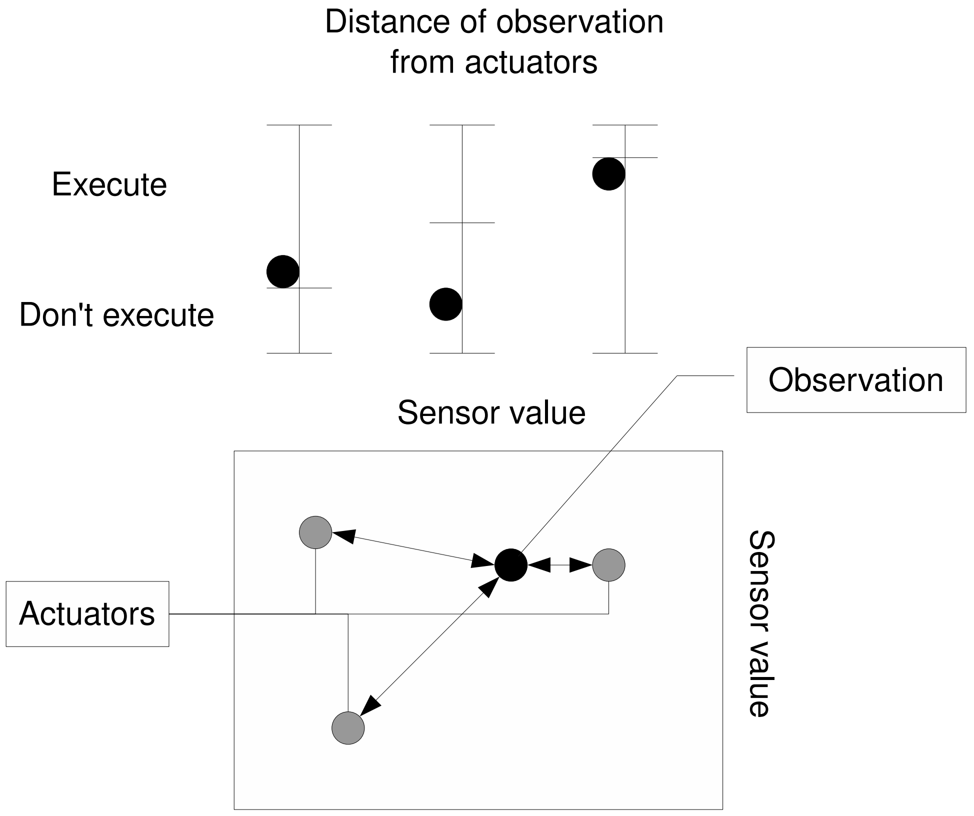
variation operators must at least follow the criteria symbolized by general formulæ (45, 46). These formulæ state that the generated solution set S' cannot be a subset of S, and the probability of generating the set S' given S is greater than generating S' given an arbitrary solution sample $\mathbb{S}$.

$$S \xrightarrow{V} S' : P(S \not\supset S') > 0 \tag{45}$$

$$P(S'|S) > P(S'|\mathbb{S}) \tag{46}$$

Where S is a set of solutions. V is the set of variation operators and $\xrightarrow{V}$ means the use of V to create a new set of solutions. Finally, $\mathbb{S}$ is an arbitrary solution set drawn from the solution space $\mathbb{N}_S$. Basically, these equations specify that the variator must contribute to the search. They present a unambiguous formulation of the necessary and sufficient conditions the variator must meet.

Figure 7: Cluster rule

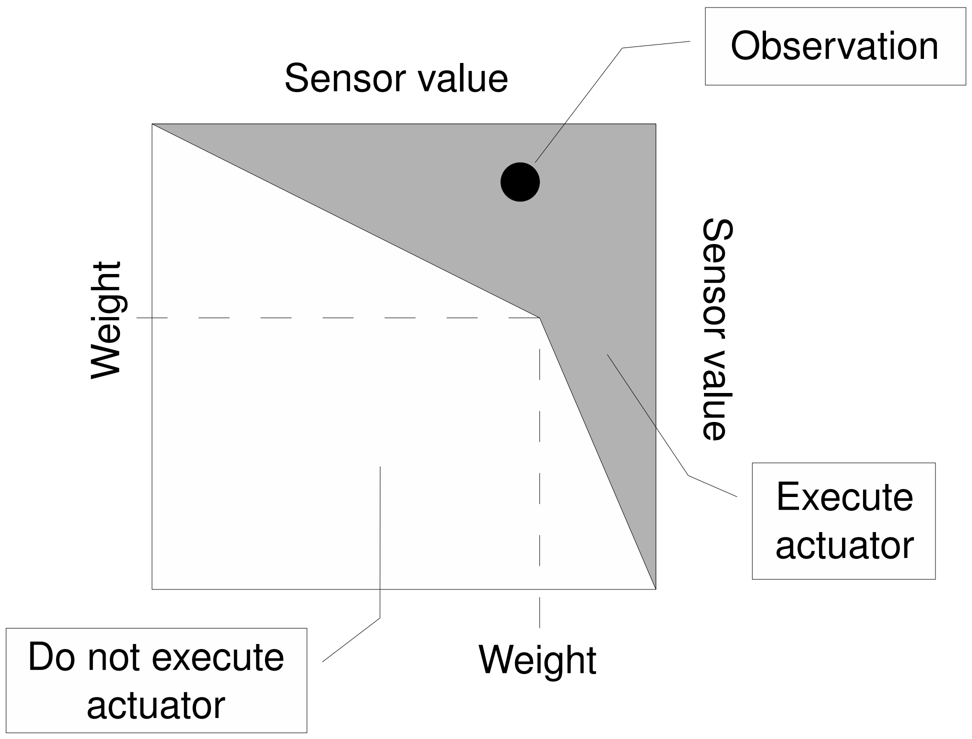


3.7.2 Agent. Each agent has a set of sensors and rules. There are a set of actuators, but only the rules can access these. See Table 9. Additionally, each agent has a memory, which it can use to store information about itself, such as its fitness value, and information used by its actuators, such as parameter values and chromosomes for creating new swarm agents.

When the simulator executes an agent, it first executes all its sensors. This sets the values in all the sensor variables used by the rules and any parameter variables necessary for the actuators, signified by "state" in the table. Then, each rule is executed, and the rule's execution is mediated by the weights, as shown in figure 5. There two main ways the weights can be used to execute the rules, as used in machine learning [23]:

- clustering, see Figure 7

Figure 8: Classification rule



- classification, see Figure 8

The cluster rule is formalized by equation (47). The classification rule is formalized by equation (48). *Only the classification rule is used in this research.*

$$\Delta = \bigcup_{\delta} \exists \mathrm{r} \in R : \mathrm{distance}(\vec{weights}_r, \vec{parameters}_{sensors}) \to \delta \tag{47}$$

$$\Delta = \bigcup_{\delta} \exists \mathrm{r} \in R : \mathrm{mean}(\vec{weights}_r. \times \vec{parameters}_{sensors}) \to \delta \tag{48}$$

If a rule is fired, it executes its actuators. The actuators are the only aspect of the agent that can affect the agent's environment as well as its state. The sensors and rules can only affect the parts of the agent's state that deals with the decision process, which is composed of sensor variables, weights, and actuator parameters.

3.7.3 Location. Like the agents, a location has both a sensor and a rule set. There are two main differences between agents and locations. First, only locations hold agents. Locations do not hold locations, agents do not hold agents, and agents do not hold locations. Second, agents' sensors are restricted to their location and their location's neighbors, and the other agents at these locations. Other than that, a location's sensor and rule set is very general, in order to allow for the widest range of possible scenarios. An example of a location rule is fitness update based on fitness of local agents.

3.7.4 Metrics. There are two primary types of metrics, runtime metrics and metrics that are evaluated once the simulation is complete. Metrics can either evaluate the whole scenario, or be responsible for evaluating only a section. Once the simulation is complete and all metrics have been evaluated, the values are placed in the objective variables, which are returned to the variator, which in turn gives them to the selector. See figure 5.

3.7.5 Scenario Actuators. The scenario actuators are meant for two purposes: global scenario actuators and general housekeeping. An example of the former is updating the graph model as the locations are changed. For instance, if a location is deactivated, the graph update actuator removes any edges connecting to the location so agents do not visit an invalid location. An example of a housekeeping actuator is a memory monitor, which ensures the agents do not cause a memory overflow.

3.8 Summary

The problem domain is formalized into a planning problem. To create an adaptive swarm, both global and local fitness functions are needed. The swarms with global and local fitness functions are represented with DEC-POMDP and I-POMDP models respectively. The inability of these models to represent arbitrary changes in the swarm and environment is discussed, and a new, general model augmentation F(*-POMDP) is introduced that allows the models to represent the mentioned arbitrary changes, as required by the SOMAS problem domain. With the assumption that the operators in the ruleset are fixed, the solution domain is simplified to the generation of the correct operands for the operators. Since it is intractable to derive solution policies from the augmented POMDP models with a local search algorithm, approximate global search algorithms are investigated.

Due to fitness landscape analysis, an evolutionary algorithm is considered best suited to generate swarms. The generation of the operands requires an architecture that can search for and evaluate the behaviors generated by the optimal or an optimum set of operands. Consequently, the architecture design is discussed.

The architecture is composed of two parts, corresponding to the two perspectives: global and local observations of environment state and reward. Finally, the algorithm domain is tied into the SOMAS architecture. The implementation of the architecture, and how it is tied into a comprehensive testing system, is discussed in the following chapter.

IV. Realization of SOMAS Architecture

4.1 Overview

With the high level design defined in Section III, the low level implementation details are developed. The general design pattern and method for developing the software architecture is covered in Section 4.2. Section 4.3 discusses the software packages available for use in implementation. The search algorithms for the SOMAS domain are discussed in Section 4.4. Section 4.5 describes how the simulation for evaluating the SOMAS agents in the network environment, described in Sections 4.6 and 4.7 respectively, is implemented. Finally, Section 4.8 explains the difficulties and ease of transitioning the simulation to a real world environment.

4.2 Software Development Methodology

The primary programming paradigm used for the SOMAS architecture is object oriented programming. This method of programming is summarized by a set of key attributes: encapsulation, loose coupling, and polymorphism. Object oriented design is used to reduce code redundancy, simplify programming, and allow for easy architecture extension. Following and generalizing these three design principles, the properties design pattern is used for the SOMAS architecture, see Section E.

The main distinctions between the properties design pattern and object oriented design is that the properties design pattern has full graph inheritance, and the classes can be defined during runtime. These two attributes make the properties design pattern more general than standard object oriented design, and allows the following design patterns to be used.

4.2.1 Shared Object Coupling with Access Mediation. The properties design pattern allows the objects to be instantiated so they are loosely coupled. However, the scenario generatior couples many of the components by causing them to share a common object. While this does couple the components in one way, it is not the same as tight coupling, since components that share the same object do not need to know anything about each others' internals. Instead, they communicate using a shared variable.

The scenario generator is the only component that directly accesses the internals of objects in order to set their parameters. In this sense, the scenario generator is termed an "Access Mediator" since once it has set the parameters on each component correctly, each component does not need to be aware of any other components. Normally, each component at least needs to know about the interface offered by a component it needs to interact with. Thus, access mediation provides an even lower level of coupling, "shared object coupling," than discussed in traditional object oriented software engineering.

4.2.2 Formal Modelling. Using lisp, it is possible to design a simple domain specific language (DSL) that completely details the property interrelationship between objects, which is called a schema. The simplified syntax allows the schema to be automatically verified. The essential conditions that must be verified are that all get operations on a variable are chronologically precceeded by a set operation, and that all set operations are precceeded by variable declaration. A property is essentially a label for a variable within a certain context. Consequently, variables can have multiple labels within multiple contexts.

The variables are labelled during the definition of a context, which is specified with the schema DSL. Additionally, the DSL specifies when the get and set operations are performed on a variable. To verify the essential condition the labels for a specific variable must be connected together to ensure there is an unbroken chain between the get and set operations.

Due to the fact that a variable can be contained in multiple contexts, and the definition of these contexts can be nested within each other, connecting up the labels can be a somewhat tricky process that requires both back and forward tracking. For instance, a variable's label series can be backtracked from a get to the original variable declaration. However, the actual set operation might take place within a context that is defined between the get and declaration. Consequently, once the backtrack has identified the original variable, a forward track is necessary to find the set operation in the nested context.

4.3 Package Selection

The needs that dictate package selection from the many possibilities covered in Section II are:

- Availability
- Ease of use
- Extensibility
- Efficiency
- Effectiveness
- Language
- Target OS
- Parallel/grid computing capabilities
- Maturity
- User community

All the following packages were selected with these criteria in mind. Not all criteria are necessarily met by the packages. For instance, many are not made for a parallel/grid environment, though they often can be adapted. But, as many criteria as possible are met by the following packages.

4.3.1 Simulation and Visualization. Since MASON is both written in Java and meets many of the requirements of the SOMAS framework, as well as being purpose built for multi agent simulation, is it the chosen simulation and visualization package. MASON is combined with JUNG to ensure the network visualization is nicely laid out. JGraphT is used in MASON for generating the random graphs used in the simulations, and for executing different graph algorithms, such as shortest path and max flow calculations. CERN's high performance mathematical library, Colt, is used in a couple areas of the SOMAS framework where probability distribution functions are required for random sampling. Weka's clustering capabilities are used to perform clustering in the self organization metric.

4.3.2 Evolutionary Algorithm. The PISA framework is chosen because it provides both a rigorous multi-objective algorithm testing environment, and many appropriate search algorithms. Even though the PISA framework [24] does not provide the general functions to implement particular variator algorithms, the sample algorithms on the PISA site are easily customized for the SOMAS problem domain. During initial development, customization of an existing algorithm is the most efficient and effective way to generate a working algorithm, compared with creating a new algorithm from scratch, so it is chosen for creating the SOMAS variator.

4.4 Search Algorithms

4.4.1 MOEA. See Section 3.6 for the discussion about the algorithms used in the SOMAS architecture and multi-objective optimization. A MOEA in the PISA framework is composed of two pieces: the variator and the selector [24]. The variator is responsible for the creation of chromosomes and their evaluation, and the selector selects the chromosomes to be used as parents for the next generation, as shown in Figure 5. Currently, the nondominated sorting genetic algorithm 2 (NSGA2) and strength pareto evolutionary algorithm (SPEA2) are the *de facto* selection algorithms for multi-objective evolutionary algorithms, and each has its advantage over the other. NSGA2 converges faster to the PF_{TRUE} than SPEA2, but SPEA2 explores the PF_{TRUE} to a greater degree [69]. Since the curse of dimensionality requires a search algorithm to be more exploratory as the number of objectives increases, increased exploitation is still required in order to find a PF_{KNOWN} significantly close to the PF_{TRUE}. The indicator based evolutionary algorithm (IBEA) [142] combines the exploitation of NSGA2 and exploration of SPEA2 by using a quality metric, such as the epsilon or hypervolume metric, and is used for selection in the thesis.

4.4.2 Operators and Parameter Selection For MOEA.

As discussed in Section 3.6.1, the fitness landscape of the SOMAS solution domain is mountainous and rough. This means there are many local optima varying greatly in height, and the slopes to the optima are not smooth.

The implemented mutation operator is uniform mutation: it looks at each bit in the chromosome and flips it according to a probability. Since multiple objectives in MOPs do not generally map in a regular pattern to the decision space, it is useful for the MOEA to be highly exploratory. For this reason the mutation probabilities are higher than in single objective GAs. Additionally, the large number of local optima with a large variance in fitness also require the search to be highly exploratory. Consequently, the probability of mutation is set to 1.0 and the probability that a particular allele is mutated is set to 0.1. The likelihood that at least one allele is mutated is $1 - 0.9^{\alpha}$, where α is the number of alleles in the chromosome. Since mutation in a GA is traditionally set to a very low value, such as around 0.01, this parameter setting is comparatively very high for a chromosome of significant length.

The crossover probability is low in regard to normal GA settings, due to the characteristic of MOPs where the good building blocks tend to have high epistasis and occur rarely [36]. The crossover operator is a uniform crossover to avoid the difficulties with high epistasis, as discussed in Section 3.6.2, where the longer a building block is the more likely it will be disrupted. Additionally, the likelihood of crossover is significantly smaller than traditionally used so as to avoid premature exploitation. The probability the crossover operator is used is set to 1.0 and the probability of crossover is 0.1 for each allele. The likelihood at least one allele is crossed over is the same as the likelihood of at least one allele mutation, which is $1 - 0.9^{\alpha}$, where α is the number of alleles in the chromosome.. Traditional GAs tend to use multi point crossover, instead of uniform crossover. So, while the likelihood of using the crossover operator is very high, the likelihood that the same number of alleles is crossed over as in a traditional GA is quite low.

4.4.3 Chromosome Evaluation. For the purposes of this investigation, the variator section of the PISA MOEA framework is decomposed even further between the variation and evaluation. Evaluation is turned into an interface, allowing any evaluation object to be used that can translate the chromosome representation used by the variator. A chromosome generates a specific behavior depending on the scenario it is evaluated within. However, its general behavior is not restricted to a particular scenario.

For example, if the chromosome is evaluated across multiple scenarios, then it has an objective set for each scenario, as well as different kinds of meta-scenario objectives, such as robustness or stability or survivability. This is because the chromosome represents weights and parameter for the same set of sensors, rules, and actuators across all scenarios.

It is possible that there are general settings for these agent elements that are more effective and efficient than any other setting, for a specific set of scenarios. Of course, according to the No Free Lunch theorem (NFLT) for optimization [139], it is unlikely that there is a particular most-fit chromosome across all scenarios. But, it is not an impossibility, since the total set of scenarios is not equivalent to the total set of fitness functions that can evaluate the chromosome, and when the fitness function domain is restricted the NFLT does not always hold [59].

Additionally, certain implementation concerns come into play depending on the number of scenarios and the number of objectives per scenario. Since the scenarios are randomized, two runs do not necessarily give the same objective values. Consequently, the chromosome has to be evaluated over multiple runs on the same scenario, with an overall statistic (such as sample mean or sample variance) representing the chromosome's fitness, or all objectives from all runs are included in the objective space. The former is simpler, but the latter is more accurate since no information is lost. However, as the number of objectives increases, the runtime of the selection algorithm increases. Depending on the algorithm used, the overhead may climb exponentially. If it does, then it is necessary to use dimensional reduction to speed up evaluation, approximate ranking, or reduce the population sizes [36]. Thus, to make the objective space manageable, a statistic of the runs is used for the objective space, instead of all objective values for all runs.

4.4.4 Single Scenario Evaluation. The swarm behaviors are evaluated directly in a scenario that simulates the setting for the swarm behavior. A swarm that is meant to stop a network attack is evaluated in a network attack scenario. A swarm that is meant to find the vital elements of a network is evaluated in a vital element identification scenario.

The objective values are calculated according to metrics that measure the performance of the swarm according to the mission objectives. For instance, in the Information War scenario, the amount of information on the network that is protected, the number of nodes that still remain, and the number of malicious agents eliminated all contribute to defining the optimal behavior of a defensive swarm.

4.4.5 Multi Scenario Evaluation. Some agent behaviors can be very complex, and good building blocks are consequently far apart and hard to construct. Consequently, it may be beneficial to use simpler behaviors as stepping stones to help the chromosome evolve towards the goal behavior. This concept is very simple to implement in a MOEA. All it means is that more objectives are added to the objective space and multiple simulations are used to evaluate the chromosome. The results are then averaged across the different runs as in the case with a single simulation.

4.4.6 Self Organization. The concept behind the self organization formula is to quantify the amount of information needed to predict the future. This information is the $\log_2$ of the number of causal states there are in a history. A causal state is a set of pasts that have the same future configuration [117].

While it is possible to use the complete information available in the system to quantify the predictive information, this can be very complex, and perhaps not necessary to get a useful metric. Consequently, the use of statistics is considered. The statistic for prediction can either be local or global. A local statistic is good for fine grained analysis, which is preferable for visualizing the self organization taking place on the network.

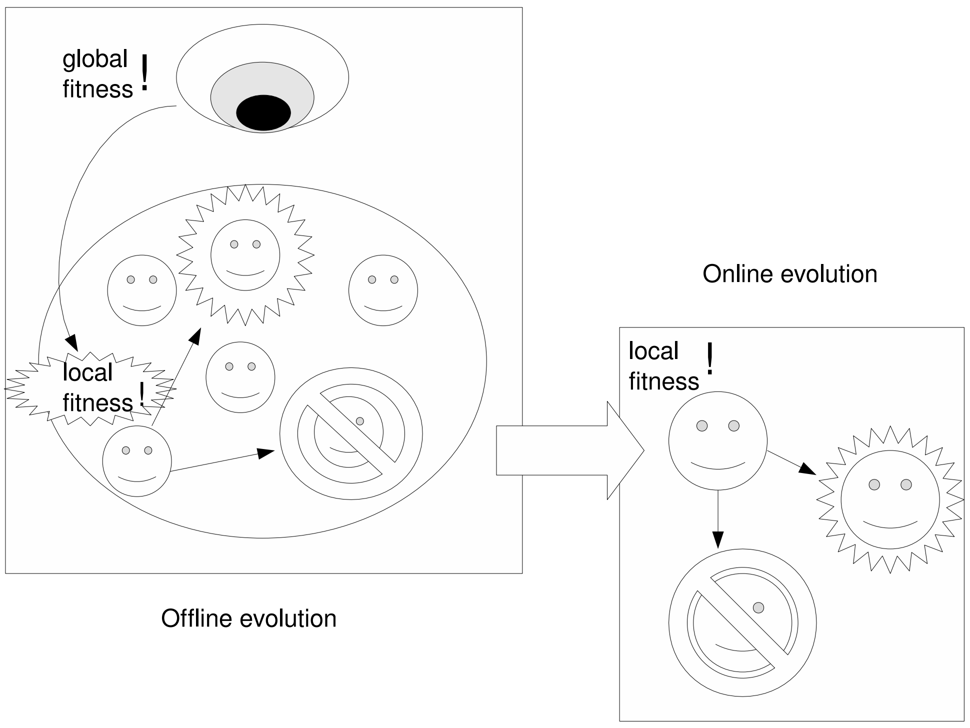
The downside of a local statistic is complexity. Since all potential influences for the extent of the measured past and future must be taken into account, the number of considered nodes grows exponentially as the length of measured time increases. Consequently, a global statistic that summarizes the information for the whole network is sufficient.

Where a property can be specified by a statically sized set of values the actual set is used, such as the fitness value for each network node since the number of network nodes does not change. Otherwise, where the set changes, such as the parameter values for each agent in a F(*-POMDP) swarm that is on a particular network node, a statistic is necessary. The statistic used in this case is the sample mean.

In most cases, the simulator is not executed for a long enough time to generate the true future distributions for a given causal state. Consequently, some kind of clustering is necessary to group similar future distributions. Otherwise, in such instances, every future is unique, and there is as many causal states as there are futures. Thus, the predictive information metric becomes meaningless.

To cluster the time sequences, expectation maximisation clustering is used. Expectation maximization clusters data with the simplest probability distribution possible to predict the greatest amount of data [23]. This is different than the technique used by [117], where a $\chi^2_{.05}$ distribution is used. Expectation maximization is used because the history does not contain enough time points to produce a large enough

Figure 9: Offline optimizing online search fitness function



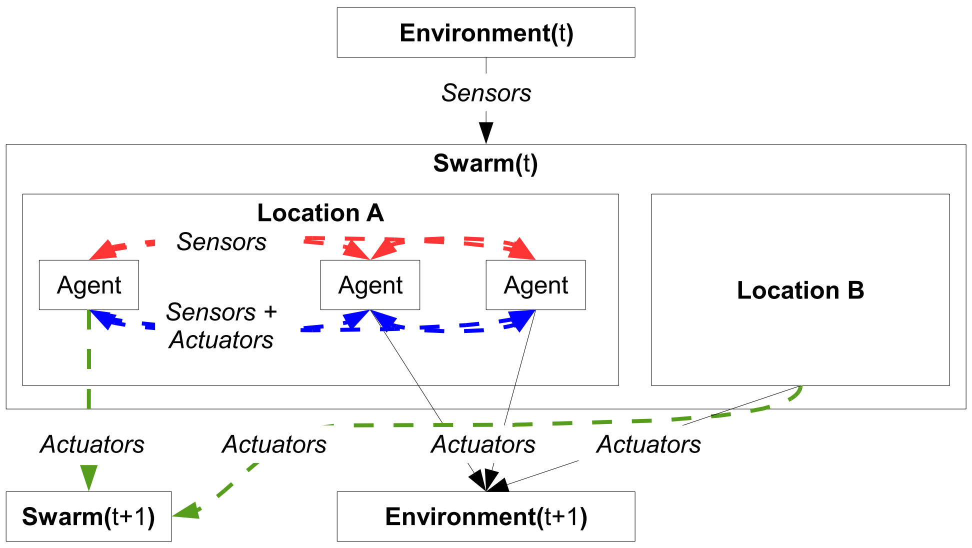
sample space for the $\chi^2_{.05}$ distribution. The basic description of the procedure is found in algorithm 3.

Algorithm 3 Self organization calculation algorithm

```
Future distributions ← $\text{cluster}_{EM}$( futures for all pasts)
Return $\log_2$(future distributions)
```

4.4.7 Entangled Hierarchy. In order for the online search to be effective, its ruleset parameters need to be optimized during the offline process so the emergent search heuristic approximates the global state as much as possible. The offline process, in general, evolves the online local fitness metrics, as shown in Figure 9. The online local fitness metrics that are evolved are the basis for the online swarm's entangled control hierarchy, along with the chromosomes that each agent carries for instantiating new agent swarm solutions. This entangled hierarchy consists of 3 mechanisms, shown by Figure 10.

Figure 10: SOMAS entangled hierarchy



- The red arrows show how the swarm concurrently communicates through sensors during a single time step t. Each agent can view the fitness value of every other agent at the local node and these fitness values are used in each agent's rules.
- As shown by the blue arrow, when the swarm has evolutionary actuators, each agent can view the chromosomes of all the other agents, and each chromosome's fitness value, for crossover.
- The green arrow shows how the swarm at t influences the swarm at $t+1$. In this case, since the swarm cannot look into the past, the influence is unidirectional. The actuators in swarm(t) that can affect swarm(t+1) are the agent creation and deletion actuators. Thus, the red control structure affects whether agents are created and deleted, and the blue control structure affects which chromosomes are used to create the new agents and what alleles make up the chromosomes.

These 3 mechanisms of control specify increasing levels of entangledness in the control hierarchy.

1. **NS** *No Search* : Red

   Temporal : local control

   Solution space : no control

SOMAS agents are only created from a single set of parameters (a single chromosome).

2. **NE** *No Evolution* : Red + Green
   Temporal : regional control
   Solution space : local control
   SOMAS agents can be created from multiple parameter sets (multiple pre-defined chromosomes).

3. **EV** *EVolution* : Red + Blue + Green
   Temporal : regional control
   Solution space : regional control
   SOMAS agents parameter sets are evolved (new chromosomes are defined)

Once the environment changes, the online search may no longer be effective. Consequently, the offline search must be capable of generating new online searches to adapt to the changing environment, or generate online searches that can adapt to a wide range of environments.

4.5 General Simulation Implementation

As discussed in Section 3.7, evaluation is accomplished by generating a SOMAS swarm from the chromosome, executing the swarm in a simulation, and using a set of metrics to generate the chromosome's objective values. This section describes the components used to construct the simulator. The SOMAS swarm and simulation must be able to represent all elements mentioned in Section 3.2. Since the elements are all self contained and do not need to directly interact with the other elements, the design is based on a component model, where each part is a component that is run independently during the simulation.

This design has a number of benefits. It allows the design and simulation to be easily extended, since each component is loosely coupled with the other components. Other frameworks such as NS2, SWARM, or swarmfare (see Section 2.7) can be used

Figure 11: Simulation component generation

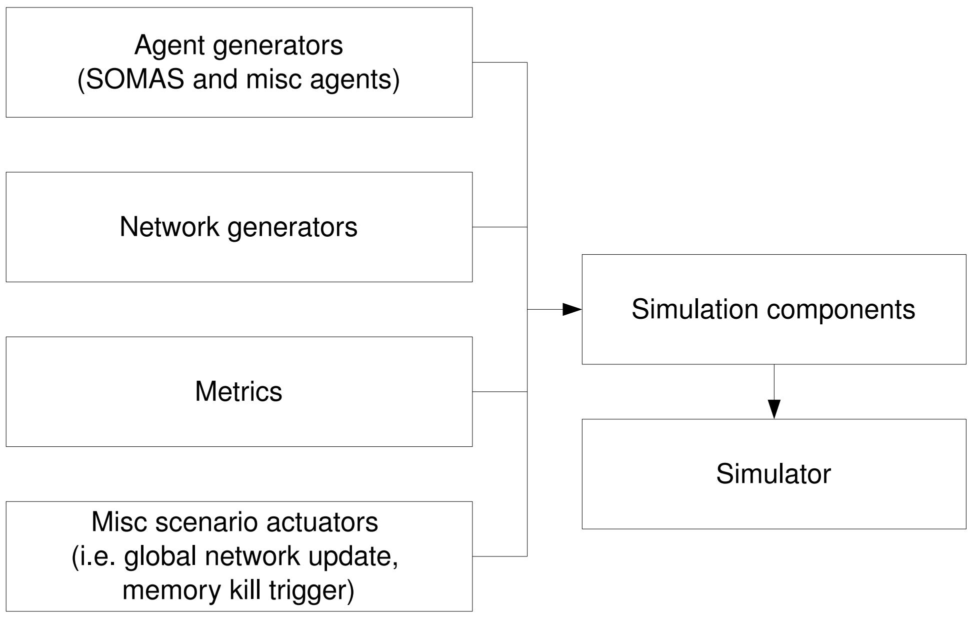


Table 9: Distinctions between agent aspects

Aspect	Input	Output
Sensor	environment	decision state
Rule	n/a or decision state	decision state
Actuator	n/a or decision state	agent state or environment

to augment the components, providing extra capabilities. Since each component can be executed independently of the others, their execution can be parallelized without the need for significant synchronization, thus speeding up the simulation execution.

The simulator is given a series of components which it simulates individually, either according to a deterministic or non-deterministic schedule. These components are the agents, vertices, metrics, and miscellaneous actuators, as in figure 11. The agents and vertices are combined to create the scenario and behavior. The metrics are used to evaluate the effectiveness of the swarm behavior in accomplishing the required objective. The miscellaneous scenario actuators are required for any other housekeeping functions that are not covered by the previous three components.

4.5.1 Simulation Components. Each iteration of the MASON simulation, see Section 2.7, executes a sequence of components. The components can either be ordered or disordered and the sequence can be randomized. In this investigation the sequence is not randomized, however, and is executed in the order that components are added during the scenario creation with the scenario generator.

There is also a priority level associated with each component, and components with a lower level are executed before components with a higher level. The execution order of components with the same priority is arbitrary. The priority level is necessary because certain kinds of components should not be executed before others. For example, the location actuators should be executed after the agent actuators, since location state affects agent rule decisions. If the execution order of the two types of actuators is unspecified some agents execute before and some after a location changes its state. Consequently, the agents that execute after will potentially react to a different location state than the first agent set.

4.5.2 Scenario Generator. The scenario generator is the heart of the simulator software. It is responsible for creating all the simulation components that are executed by the simulator, which includes the use of sub-generators. Figure 11 details the components that the scenario generator creates and adds to the simulator. The primary responsibility of the scenario generator is to ensure all the components share the correct variables. Most components share a variable dependency, such as the rules, which are dependent on the sensor variables that the sensors set. The scenario generator either creates and distributes these variables, or calls another generator to create the necessary variables.

4.5.3 Agent Generator. The agent generators are necessary because the scenario generator cannot create all agents that the simulation needs. During the runtime, while most of the simulation entities' existence is constant, the agents' existence is not. The agents, locations, and miscellaneous actuators all have the potential to create or delete agents. Consequently, runtime agent generators are necessary as well. All agent generators have a common dependency on 3 variables. These variables contain the simulation's scheduler and descheduler, and the agent's location. Besides these 3, each agent generator has its own specialized variable dependencies. The unevolving agent generators are very unspecified in the additional variable dependencies. They may have no more dependencies or many more dependencies. The evolving agents generators require one more kind of variable, the chromosome, which is used to instantiate the agent's phenotype.

4.5.4 Network Generator. The network generator is primarily responsible for composing the network topology. Since the topology does not change during the execution, though nodes and links may be deactivated, the topology can be generated entirely before the simulation is executed. Thus, the network generator is part of the scenario generator.

4.5.5 Memory Kill Trigger. During preliminary testing, there was no limit to the number of agents that could be created. Certain chromosomes would create agents until the Java Virtual Machine (JVM) crashed. To stop this from happening, the memory usage during the simulation is monitored, and the simulated is stopped if the memory usage passes a certain threshold. Chromosomes that cause the memory threshold to be exceeded are considered infeasible.

4.5.6 Graph Update. A number of the agent sensors and actuators depend on a regional view of the graph. This way agents can determine the best location to travel to, or the best location to deactivate. The graph updater keeps track of what locations and links have been deactivated, and updates the global view of the graph accordingly.

4.6 SOMAS Agent Implementation

A SOMAS agent is composed of 3 basic elements: sensors, rules, and actuators. The rules use a decision function on the sensor values to decide whether to execute

Table 10: All implemented sensors, rules, and actuators

Sensors	Rules	Actuators
Agent List	Comm Agent	Change Location
Agent Order	Compromise Agent	Compromise
Agent	DDoS Agent	Compromise Agent Info Exchange
Agent With Most Pheromone	Execute All	Compromise Location
ArcTan Filter	Receiver Agent	Create Agent
Compromised Node	Sender Agent	Create Agent
Compromise	Weighted	Crossover Local and Location Chromosome
Context Pheromone		DDoS Comm Agent Info Exchange
Local Chromosome Order		Decrease Pheromone
Local Chromosome With Most Pheromone		Delete Agent
Location Agents Cumulative Pheromone		Delete Agent List
Location Chromosome Order		Deposit Bad Info
Location Chromosome With Most Pheromone		Increase Pheromone
Location Marked		Increment Good Info
Location Number Agents		Increment Info
Location Number Neighbors		Mark Location
Location Order		Mutate Self Chromosome
Location Pheromone		Mutation Operator
Location		Null
Location With Most Pheromone		Send Comm Agent List
Metric Aggregator		Set Location Pheromone
Next Hop		Set Pheromone On Self Chromosome
Next Network Search Hop		Unmark Location
Property		
Random Value		
Receiver Node		
Select Thing		
Self Pheromone		
Sender Node		
Target		
Thing Pheromone Order		
Thing With Most Pheromone		

their actuators. The range of possible sensors, rules, and actuators in an agent is detailed in Table 10.

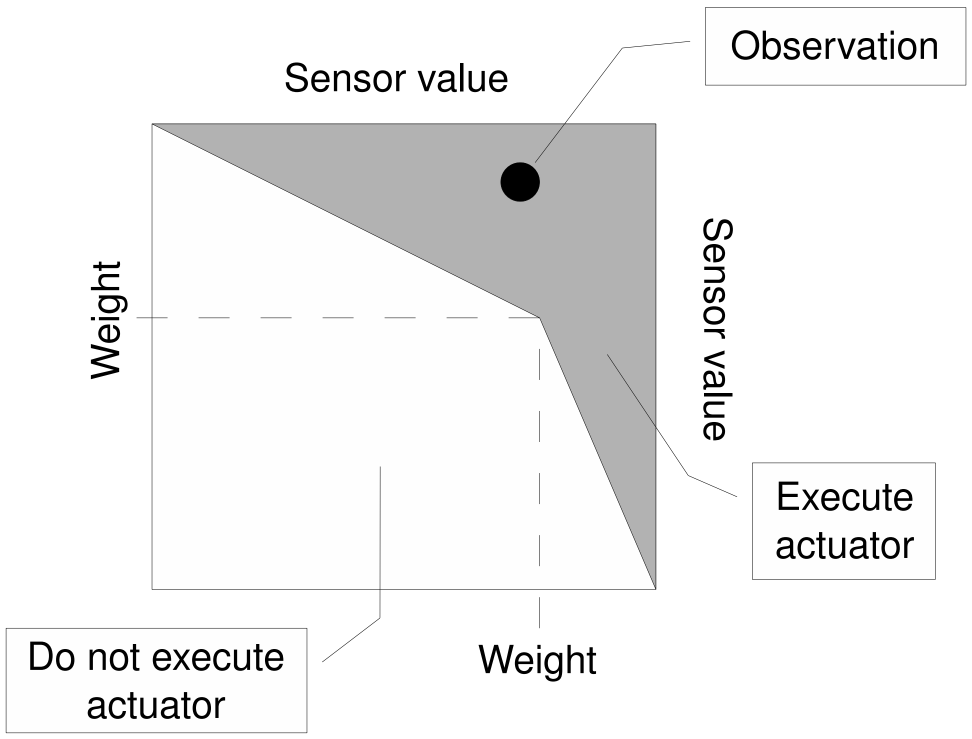
There are two steps to the execution of an agent component. First, its list of sensors is executed. Then its list of rules is executed. This ensures that all necessary sensor values are initialized before they are used by the rules.

There are two ways the sensors, rules, and actuators are specified in an agent. In the non-searching agents, the parameters are all set, consequently the relation between the different aspects are all hardcoded. The sensors, rules, and actuators are still differentiated, but the actuators are accessed directly instead of indirectly through datastructures. On the other hand, the relationships between the sensors, rules, and actuators are based on searching parameters within the searching agents. Thus, they are accessed indirectly through a list datastructure, like the sensors and rules. The rules are responsible for specifying the decision relationship between the rules and the actuators, which changes based on the kind of decision making the agent does, i.e. simple weighted or clustering.

Besides the generic agent elements of sensors, rules, and actuators, each SOMAS agent also has a chromosome store. The chromosome store contains the set of chromosomes that are varied by its genetic operators and used to create new agents. Each chromosome has an associated fitness value, which the SOMAS agent can set. The fitness value is used to select chromosomes for variation and creation. Each SOMAS agent can see the chromosome set of all other SOMAS agents in its location.

Besides the chromosome fitness values, all agents and locations also have fitness values. The agents' fitness values can be set by themselves and by SOMAS agents. The fitness value on an agent influences a SOMAS agent's agent related rule decisions that choose an action to take, such as whether to delete a local SOMAS agent. A location's fitness value is an aggregate of the fitness values of the agents currently residing at the location.

Figure 12: Classification rule



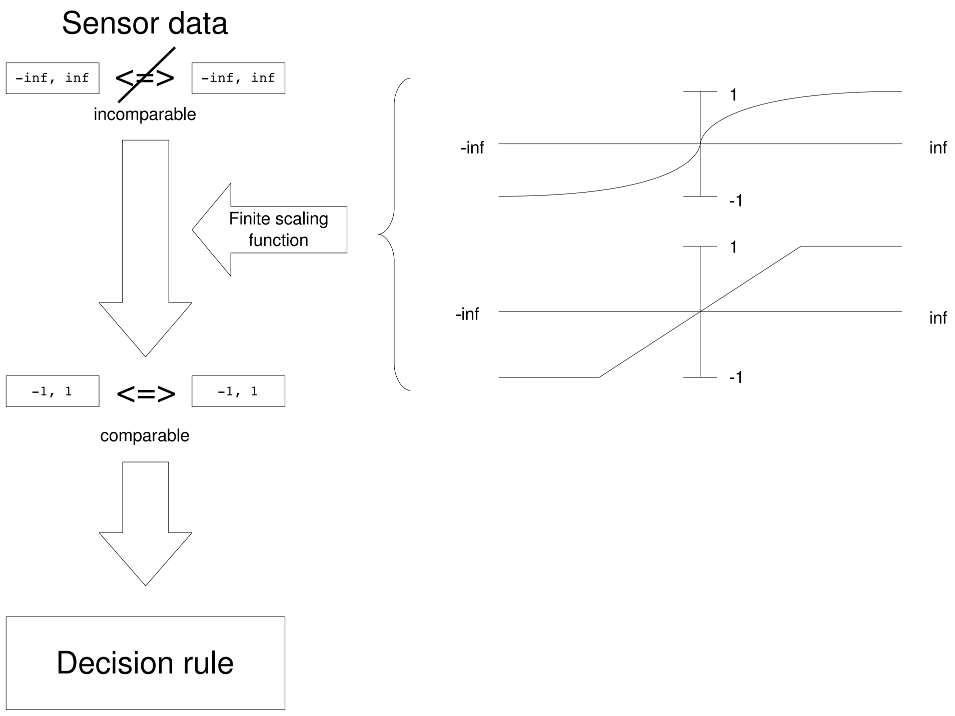
4.6.1 Basic Weighted Rule. Each rule is an activation function. These functions all follow the same format, which is analogous to a classification in machine learning [23].

$$\text{measure}(\vec{info}, \vec{weights}_{actuator}) > 0 \rightarrow actuator.\text{execute} \tag{49}$$

Where $\vec{weights}_{actuator}$ is a particular set of weights for the given actuator in the range (-1, 1), which is also statically sized. The execution function used in this implementation is to take the weighted average of the input variables. If the average is above an execution threshold, then the rule fires its actuator. As shown in Figure 6, the weights partition the sensor space into an execution and non execution region.

$$\text{convert}_{weighted}(\vec{data}) = \text{mean}(\vec{weights}_{actuator}. \times \vec{data}) \tag{50}$$

Figure 13: Sensor data transform



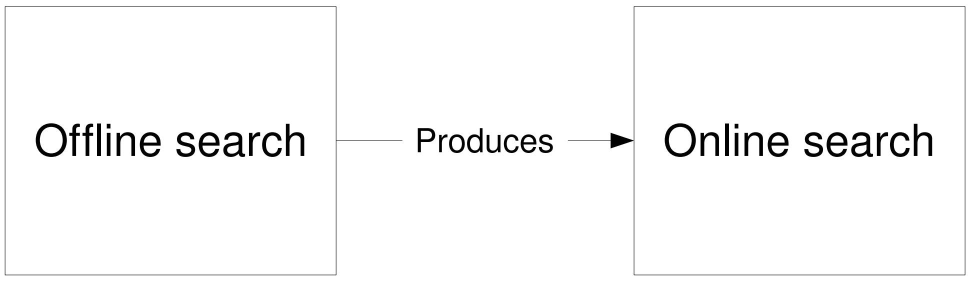
4.6.2 Sensors. Sensors convert observation data into the information the rules use to make decisions. An observation can contain data from both the agent and its environment, which includes other agents. Adding random and static values to an agent allow these value types to influence the decision rules. The formalization of a sensor is:

$$\text{convert}_{actuator}(\vec{data}) \rightarrow \vec{info} \tag{51}$$

Where $\vec{data}$ is statically sized sequence of values in the range (-inf, inf). $\vec{data}$ is a statically sized sequence of values in the range (-1, 1).

Since the data that the sensor can perceive is potentially infinite, and the different data elements that make up an observation need to be comparable to each other, the sensor requires a scaling mechanism in order to restrict the data to a finite range. This can be done in a number of ways, such as by restricting the perceivable data to a certain numerical range, or by using a trigonometric function to scale the data. Both approaches are illustrated in figure 13.

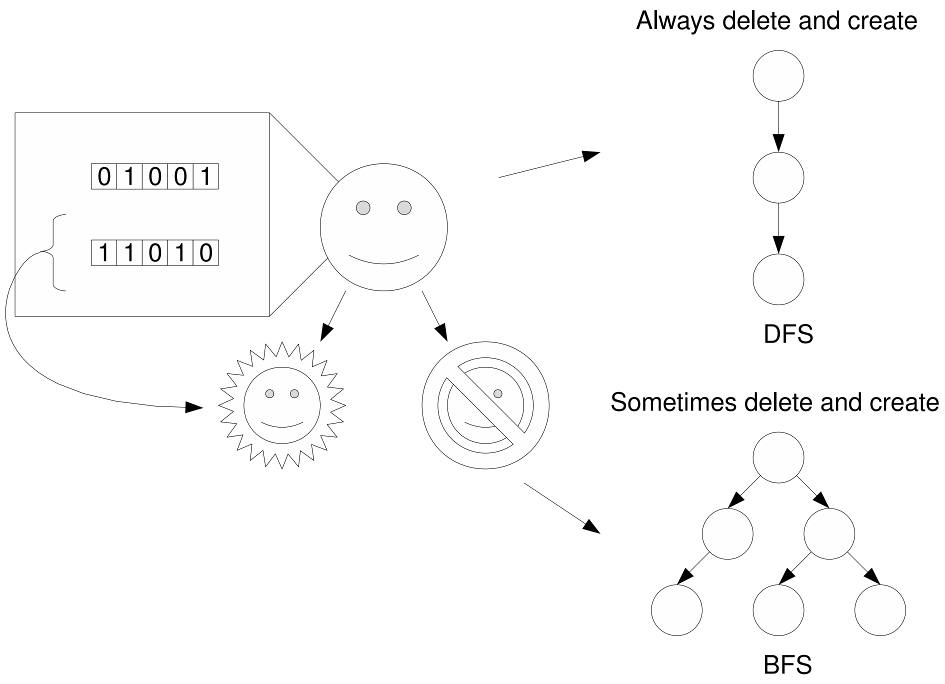
Figure 14: Online evolution



4.6.3 Online Search. The offline evolution evolves the agents local fitness function, as well as the other necessary parameters and weights. See figure 9. By using the global fitness, the offline evolution can approximate its evaluation at the local level, providing the online evolution with the ability to respond to a wider variety of changes in the environment. The local fitness approximation and action of the swarm are necessary for online search. The fitness value approximation operates by setting a local fitness function on an entity, whether that entity is the agent itself, a chromosome it carries, etc., and using the local fitness functions to fire its rules, select new agent chromosomes for agent creation, and delete other agents. The agent uses the local fitness values in a population of chromosomes to select a chromosome to crossover with its own chromosomes. It also mutates its own chromosomes at a given interval. The operators it uses for crossover and mutation are the same as those used for offline evolution.

The online search is essentially a search for appropriate agent behavior, where the search is part of the behavior. The creation and deletion of agents serves to create a hybrid between depth first and breadth first search. If an agent continually creates and deletes an agent each iteration so that only a single agent is left from the procedure, then this is equivalent to a depth first search. Alternatively, when the agent creates more agents than it deletes, it is searching along multiple possible swarm behaviors. When it deletes more agents than it creates, then the agent is performing a heuristic search, i.e. reducing the options available on the open list as described in [88].

Figure 15: Online search



The fitness landscape of online search is dynamic at two levels. At the simple level, it is dynamic because the scenario the swarm is in is dynamic. The swarm is thus searching for a behavior that best attains desired goal states. However, the fitness landscape becomes more complex when the agent activities impact the scenario and thus the state transition and reward functions. In this case, the online search landscape is represented by a transition function that gives a probability distribution over one fitness landscape transitioning to another based on evolved agent behaviors.

4.7 Network Implementation

For the purposes of the implementation, there is no difference between the agent container, see Section 3.2, and network location, see Section 3.7.3. Consequently, both are represented by the same data structure. Since the individual details of each host are not considered, the sensors and actuators for the hosts are not implemented. Instead, the sensors and actuators provided by the container are stored in the agent's data structure, for simplicity of implementation. Due to the similarity between locations and agents, each location also has a sensor and rule list.

Additionally, a location contains a neighbor list and an agent list. The neighbor list is continually updated during a simulation run, in order to take account of the changes in the network structure, such as nodes being deactivated or reactivated. This update is accomplished by the graph updater component, see Section 4.5.6.

4.8 Real World Implementation

Since the SOMAS is entirely implemented within the JVM, transitioning the system itself to a real world setting is possibly trivial. The only augmentation necessary is to distribute the components across a number of JVMs, such that each JVM has a single network location, corresponding to the real network location on which the JVM resides. Using remote method invocation (RMI), distributing the components is straightforward. Remote method invocation allows local processes to invoke and receive results from non-local objects.

Once the SOMAS is active on a new network, the number variables affecting the swarm's effectiveness increases due to the added constraints of real world networks. A partial list of these variables follows.

- bandwidth
- latency
- heterogenous host systems (hardware, OS, other applications)
- interaction of external world with network
- interaction with other multi agent systems
- trust within and between computer systems
- mission specific needs
- limited computational resources

- implementation medium (i.e. if agents were to become embedded hardware)
- highly sensitive, volatile locations
- lack of containers on target hosts

The effect of the variables on the SOMAS' environment is to increase its dynamic and heterogeneous nature. Thus, the swarms need to be given operators that increase their adaptive effectiveness.

4.9 Summary

To start the discussion of implementation the methodology for the software development is covered. Besides the standard object oriented design, a design pattern called the properties design pattern is used which allows objects to be loosely coupled and formally analyzed. The selection of packages used for implementation follows. The search algorithm selection, parameter settings, and implementation are then explained. The primary part of the SOMAS architecture, the simulation used for chromosome evaluation, is discussed next, since it is part of the variator in the search algorithm. Finally, there is a discussion of the feasibility of transitioning the simulated swarm to a real network, with the attendant increase in variables that affect the swarm's behavior effectiveness. The testing and analysis procedures and setup for deteriming whether the SOMAS architecture successfully creates desired swarm behavior is in the next chapter.

V. Measuring SOMAS Capabilities

5.1 Overview

This chapter defines and explains the testing hypotheses and methodoloy used to accomplish the research goal and objectives with the implementation from Chapter IV. The test hypotheses are described in Section 5.2. The methodology for conducting the tests is described in Section 5.3. The relevant parameters for the MOEA and simulation are in Section 5.4, and the behaviors and scenarios to be tested are covered in Sections 5.5 and 5.6.

5.2 Test Hypotheses

From Chapter 1, the research goal is to *investigate network security real-time performance using a swarm of autonomous self organized agents that evolve a non-hierarchical entangled cyberspace security management structure.* See Section 1.2.

Therefore, the primary testing objective is to determine the impact of self organization and entangled hierarchies on generating effective swarm behaviors in a variety of security scenarios. Since all the swarms are evolved using the self organization metric, a plot of the PF_{KNOWN} of each scenario is examined to visually identify the relation between the self organization objective value and the other objective values.

The success of online evolution compared to unevolving online search and lack of online search, is evidence for the significance of entangled hierarchies for swarm adaptation. As explained by Section 4.4.7, online evolution is an entangled hierarchy since each agent communicates with other local agents through its fitness value in order to determine which chromosomes to use for crossover. If evolution is not employed, then the impact of communication on swarm adaption is lessened as well as the degree of entangled hierarchy, since the swarm adaptability is limited by the agents' chromosome set. Finally, the least influential entangled hierarchy exists in the case when agents do not perform an online search.

Table 11: Hypotheses

Objective	Hypothesis
Evaluate feasibility of self organization for effective accomplishment of desired behaviors	- Self organization metric improves effectiveness of search - Visually identifiable self organization is evolved
Evaluate feasibility of entangled hierarchies for effective accomplishment of desired behaviors	- An entangled hierarchy becomes more effective as the scenario becomes more complex

Table 12: Behavior experiment summary

Behavior Name	Scenario Name	Topology Complexity	Topology Dynamics	Scenario Agent Dynamics
Minimal activity	all	n/a	n/a	n/a
Self organization	all	n/a	n/a	n/a
Preserve network	all	n/a	n/a	n/a
Enemy avoidance	EnemyAvoidanceScenario	Homogeneous	Static	Static
Intrusion elimination	IntrusionEliminationScenario	Homogeneous	Static	Dynamic
Network defense	DDoSScenario	Homogeneous	Dynamic	Dynamic
Competition	CompetitionScenario	Homogeneous	Dynamic	Dynamic
Information defense	InfoWarScenario	Heterogeneous	Dynamic	Dynamic

Table 13: SOMAS MOEA Parameters

Parameter	Value
Allele mutation likelihood	0.1
Allele crossover likelihood	0.1
Chromosomes length (bits)	5000
Evaluations per chromosome	30
Simulation iterations	100
Memory threshold	50%
Initial population	100
Mu	50
Lambda	50
Generations	10
Runs	5

5.3 Testing Methodology

There are three elements required to test the effectiveness of the SOMAS swarms.

- A variety of behaviors and scenarios to compare swarm capabilities. The behaviors are described in Section 5.5 and the scenarios are described in Section 5.6.
- Metrics for measuring and comparing the capabilities of different kinds of swarms for optimizing results for different behaviors. The metrics are described in Section 2.9.1.
- Statistical tests to accept or reject the experiment hypotheses based on the measurements. The tests are described in Section 2.9.2.

Once the SOMAS runs are complete, a series of bash scripts are used to format, analyze, and visualize the data. First, all the objective vectors are normalized and filtered, retaining only the PF_{TRUE} approximation sets for each scenario involved. Second, the metrics and statistical tests are generated. The metrics consist of the epsilon, hypervolume, and R_2 metrics, discussed in section 2.9.1. The statistical tests consist of the Fisher exact and the Mann-Whitney non-parametric tests, discussed in section 2.9.2. Finally, the Pareto front and the test results are converted to graphical format for ease of reading and analysis.

The PF_{KNOWN} for each generation is derived using the PISA dominance ranking function. Gnuplot is used for plotting the PF_{KNOWN}. All permutations of objective functions in sets of 3 are graphed in 3D scatterplots. The shade of subsequent EA generations in the plots are darker so as to show the EA's search progression during its execution. Besides the scatterplots, select chromosomes are executed and visualized with the MASON simulation to determine whether recognizable self organization is produced.

The results of the statistical tests are plotted with R using a shaded grid, where grid cells represent test hypotheses and darker shades correspond to lesser a values for rejecting the null hypothesis. Complete black entails a statistically significant result, allowing a reader to quickly guage how a particular scenario fares in the tests.

Statistical significance means the a value for the experiment must be less than 0.05 in order to reject the null hypothesis.

5.4 Experiment Settings

The population size for each generation is 100, where 50 members come from the previous generation and 50 members are produced by the variator. The initial population consists of 100 individuals. Each algorithm is run for 10 generations to produce a total of 1100 individuals, and 33000 simulation evaluations, due to the use of 30 simulation runs per individual, see section 2.9.2. These parameters are chosen so that a run takes at most 2 days to complete, while still providing enough results to run the statistical tests and generate dependable p-values. If memory consumption after an iteration passes 50% of the maximum heap size, then the simulation is halted and the chromosome's infeasibility objective is set.

5.5 Swarm Behaviors

As explained in Section 3.2 there are two primary elements to the problem domain: SOMAS agents and the environment. The environment consists of the network and the non-SOMAS agents (scenario agents). Each part of the environment contributes to the environmental complexity.

- The network can either have a homogeneous or heterogeneous layout. A homogeneous layout means that nodes are connected with uniform probability, with the qualification that each node pair only has one link. A heterogeneous layout means that the probability of node connection is non uniform.
- The network topology can either be static or dynamic. A dynamic topology does not retain the same node and link configuration throughout the scenario. In these experiments, a dynamic network means that a node and its connected links can be deactivated and reactivated.

- The scenario agents can either be static or dynamic. If the agents are dynamic, then their locations, parameters, and the total number of agents can change.

The following subsections present a range of formalized behaviors that can be tested in the network environments. However, the following behaviors are not tested due to time constraints.

1. Network preservation
2. Vital vertex identification
3. Intrusion detection
4. Network attack

The behaviors in the following list are tested in the scenarios according to Table 12.

1. Enemy avoidance (assessing intrusions for avoidance)
2. Intrusion elimination (assessing intrusions for targeting)
3. Network defense (engaging intrusions)
4. Competition (assessing, targeting, and engaging intrusions)
5. Information defense (assessing, targeting, engaging, and reacting to intrusions)

Accordingly, the scenarios in table 12 become increasingly more complex. The scenarios for Behaviors 2-4 all require many of the sub-behaviors necessary for information defense, as discussed in Section 6. Finally, the scenario for Behavior 5 is the most complex of all the scenarios, as is expected given that many of the others only require its sub-behaviors. Thus, the scenarios represent 3 levels of complexity.

The increase in complexity allows greater exploration of the capabilities of self organization and entangled hierarchies, fulfilling the test objectives outlined in section 5.2. The objectives relate directly back to the original goal in section 1.2.

By using these network security scenarios and behaviors, the metrics and tests validate or invalidate the effectiveness of self organization and entangled hierarchies for network security. Each of these behaviors is measured by a set of metrics as explained in Section 4.4.4. Each metric is a component in the simulator, see Section 4.5.1, and is loaded into the scenario using the scenario generator, see Section 4.5.2.

The behaviors are formalized in the following sections. See Section 3.2 for in depth symbol notation explanations. Subsection 5.5.1 explains the behaviors that are used by SOMAS swarms in all scenarios. Subsection 5.5.2 explains the behaviors that are only used by the SOMAS swarm in certain scenarios.

5.5.1 Swarm Specific Behaviors. The self organized behavior metric is the most important swarm specific behavior. It is directly relevant to the research goal of investigating the effect of self organization on behavior optimization.

The activity minimization and network preservation metrics are of general utility. The feasibility metric is essential in scenarios where the number of agents can increase arbitrarily. If the swarm fills up the memory space of the virtual machine, then the simulation crashes and no measurements are taken.

1. *Self Organized:* The causal states are calculated with the algorithm detailed in [117]. According to Shannon's information theory, the log_2 of the number of causal states represents the amount of information in the set of causal states.

$$\frac{1}{\log_2(|\mathbf{I}^-{}_{\vec{H}}|)} \tag{52}$$

$\mathbf{I}^-$	The set of causal states, see Section 2.5.4 for details.
$\vec{H}$	The simulation history.

The function is inverted so its maximization produces a minimum value.

2. *Non Disruptive:* The objective is to minimize agent activity on the vertices, which entails minimizing agent movement, creation, and deletion. The function measures the cumulative activity during the entire simulation run by summing

the absolute difference in agent population on each vertex for each time step.

$$\sum_{t \in T_{t>1}} \sum_{C \in \mathbf{C}} \mathrm{abs}(|\mathbf{A}_C(t)| - |\mathbf{A}_C(t-1)|) : \mathbf{A}_C(t) \equiv \mathbf{A} \cap C_S \cap S_S(t) \tag{53}$$

T	The set of all simulation time steps.
C	The set of all network vertices during the course of the simulation.
A	The set of agents.
$S_S(t)$	The scenario state at time t.

3. *Network Preservation:* The SOMAS agents can use node deactivation actuators. Deactivating nodes can coincide with achieving objectives, but deactivating the whole network is not generally a good security solution. To counteract this tendency, the number of deactivated nodes at the end of the simulation is minimized.

$$|\mathbf{C}^i(f)| : \mathbf{C}^i(f) \equiv \mathbf{C}^i \cap S_S(f) \tag{54}$$

$\mathbf{C}^i$	The set of inactive containers.
$S_S(f)$	The scenario state at the final time step.
$\lvert \ldots \rvert$	Denotes set cardinality.

4. *Feasibility:* Feasibility is included in the behavior list because it is one of the objective values, even though it is not a behavior. The objective measures the swarms feasibility by flagging whether the swarm passed a certain memory threshold during a simulation step. If the swarm uses too much memory, it can cause the container running the agents to crash.

$$\mathrm{test}\left(\frac{\mathrm{m}(\mathit{used})}{\mathrm{m}(\mathit{total})} > c\right) \tag{55}$$

test	Boolean function that returns 1 if the condition is true and 0 otherwise.
m	Memory measure.
c	Constant threshold value.

5.5.2 Scenario Specific Behaviors.

1. *Vital Vertex Identification:* The behavior is to identify the vital vertices in a network. The vital vertices are the minimal set of vertices in a network that cause the greatest network degradation when they are removed. The extent of network degradation is measured by comparing the cumulative length of the shortest path for all pairs of vertices and the number of disconnected vertices before and after the vertices are removed.

   This is almost the same as the metric of network degradation in [19]. The difference lies in counting the number of disconnected vertices. The original metric did not do this, and only has a single objective. Using a single objective raises the problem of rating the cost of disconnected vertices. However, making the metric produce a double objective measure eliminates the ambiguity.

   Finding the vital vertices in a network is an NP-Hard problem [19]. The difficulty of the problem is increased by the fact that the problem information is only partially observed, each agent only has a partial view of the whole network. Equations (56-57) measure the network degradation. Equation (58) measures how concisely the agents can identify the vital vertices.

$$\sum \forall u \forall v (\text{shortestpath}(u, v) : u, v \in \mathbf{C} \cap \mathbf{S}_S(f) \wedge u \neq v) \tag{56}$$

$$\sum \forall u \forall v (\text{pathexists}(u, v) : u, v \in \mathbf{C} \cap \mathbf{S}_S(f) \wedge u \neq v) \tag{57}$$

$$|\mathbf{ID}(f)| : \mathbf{ID}(f) \equiv \mathbf{ID} \cap \mathbf{S}_S(f) \tag{58}$$

$\mathbf{C}$	The container set.
$\mathbf{ID}$	The set of vertices identified as being vital vertices.
$\mathbf{S}_S(f)$	The scenario state at the final time step.
$\lvert \ldots \rvert$	Denotes set cardinality.

2. *Intrusion Detection:* Intrusion elimination consists of maximizing the number of correctly identified nodes, Equation (61), and minimizing the number of incorrectly identified nodes, Equation (62).

$$\frac{1}{|\mathbf{ID}_{\surd}|} : \mathbf{ID}_{\surd} \equiv \mathbf{ID} \cap \mathbf{C}^c \cap \mathbf{S}_S(f) \tag{59}$$

$$|\mathbf{ID}_X| : \mathbf{ID}_X \equiv \mathbf{ID} \cap \mathbf{C}^u \cap \mathbf{S}_S(f) \tag{60}$$

$\mathbf{C}^c$	The set of compromised containers.
$\mathbf{C}^u$	The set of uncompromised containers.
$\mathbf{ID}$	The set of containers identified as being compromised.
$\mathbf{S}_S(f)$	The scenario state at the final time step.
$\lvert \ldots \rvert$	Denotes set cardinality.

3. *Enemy Avoidance:* Enemy avoidance consists of maximizing the number of safe SOMAS agents, Equation (61), and minimizing the number of unsafe SOMAS agents, Equation (62). While it may seem that these two objectives are just the inverse of each other, that is not the case when the SOMAS swarm can increase or decrease the number of SOMAS agents.

$$\frac{1}{|\mathbf{A}^s|} : \mathbf{A}^s \equiv \mathbf{A} \cap \mathbf{C}^c \cap \mathbf{S}_S(f) \tag{61}$$

$$|\mathbf{A}^u| : \mathbf{A}^u \equiv \mathbf{A} \cap \mathbf{C}^u \cap \mathbf{S}_S(f) \tag{62}$$

$\mathbf{C}^s$	The set of safe containers.
$\mathbf{C}^u$	The set of unsafe containers.
$\mathbf{A}$	The set of SOMAS agents.
$\mathbf{S}_S(f)$	The scenario state at the final time step.
$\lvert \ldots \rvert$	Denotes set cardinality.

4. *Intrusion elimination:* The intrusion elimination behavior is similar to the competition scenario in Section 6. The difference is that the enemy agents do not try to destroy the network. Instead, the compromise agents merely compromise and deposit inactive DDoS agents. The SOMAS swarm's behavior is to make the number of compromised nodes as small as possible.

$$|\mathbf{C}^c(f)| : \mathbf{C}^c(f) \equiv \mathbf{C}^c \cap \mathbf{S}_S(f) \tag{63}$$

$\mathbf{C}^c$	The set of compromised containers.
$\mathbf{S}_S(f)$	The scenario state at the final time step.
$\lvert \ldots \rvert$	Denotes set cardinality.

5. *DDoS defense:* There is only a single objective for this behavior, which is to maximize the number of targetted nodes that are active at the end of the simulation.

$$\frac{1}{|\mathbf{C}^{at}|} : \mathbf{C}^{at} \equiv \mathbf{C}^a \cap \mathbf{C}^t \cap \mathbf{S}_S(f) \tag{64}$$

$\mathbf{C}^a$	The set of active containers.
$\mathbf{C}^t$	The set of targeted containers.
$\mathbf{S}_S(f)$	The scenario state at the final time step.
$\lvert \ldots \rvert$	Denotes set cardinality.

6. *Swarm Competition:* The behavior is to both delete all enemy agents and preserve all friendly agents. The blue (friendly) objective function is in Equation (65). It states that the cumulative number of SOMAS agents deleted during

the scenario run should be minimized. The red (enemy) objective function is in Equation (66). It states that the cumulative number of enemy agents deleted during the scenario run should be maximized.

$$\frac{1}{\sum_{t \in \mathbf{T}_{t>1}} |\mathbf{A}^b(t)|} : \mathbf{A}^b(t) \equiv \mathbf{A}^b \cap \mathbf{S}_S(t-1) - \mathbf{S}_S(t) \tag{65}$$

$$\sum_{t \in \mathbf{T}_{t>1}} |\mathbf{A}^r(t)| : \mathbf{A}^r(t) \equiv \mathbf{A}^r \cap \mathbf{S}_S(t-1) - \mathbf{S}_S(t) \tag{66}$$

$\mathbf{A}^b$	The set of friendly agents.
$\mathbf{A}^r$	The set of malicious agents.
$\mathbf{S}_S(t)$	The scenario states at time t.
$\|\ldots\|$	Denotes set cardinality.

7. *Network defense:* The basic behavior is to maximize the amount of friendly information delivered to friendly nodes, Equation (67), and minimize the amount of friendly information sent to malicious nodes, Equation (68). All information is friendly for the purposes of the scenario.

$$\frac{1}{\sum_{t \in T_{t>1}} |\mathbf{i}^B(t)|} : \mathbf{i}^B(t) \equiv \mathbf{i} \cap \mathbf{C}_S^B \cap \mathbf{S}_S(t) - \mathbf{S}_S(t-1) \tag{67}$$

$$\sum_{t \in T_{t>1}} |\mathbf{i}^R(t)| : \{\mathbf{i}^R(t) \equiv \mathbf{i} \cap \mathbf{C}_S^R \cap \mathbf{S}_S(t) - \mathbf{S}_S(t-1) \tag{68}$$

$\mathbf{i} \in \mathbb{A}$	The set of information packets
$\mathbf{C}_S^B$	The set of friendly receiver containers' states.
$\mathbf{C}_S^R$	The set of malicious receiver containers' states.
$\mathbf{S}_S(t)$	The scenario states at time t.
$\|\ldots\|$	Denotes set cardinality.

Table 14: Agent types

Type	Sensors	Rules	Actuators
SOMAS	- Chromosome fitness - Neighbor fitness - Local agent fitness - Location and neighborhood aliveness	- Weighted rules	- Evolve selected chromosome - Create agent from selected chromosome - Delete agent - Change location - Change self fitness - Make self chromosome have self fitness value
Defense	- Local agent types	- Execute actuators based on types of local agents - Execute rest of actuators	- Delete agents
Compromise	- Local agent types - Whether location is compromised	- If location is uncompromised then compromise it and execute compromise actuators - Execute actuators based on types of local agents - Execute rest of actuators	- Compromise location - Create DDoS agent - Steal information - Corrupt information - Exchange information with malicious agents
DDoS	- Local agent types	- If local agent has DDoS target then read target information - Execute rest of actuators	- Send info packets to DDoS targets
DDoS packet	- Next hop to destination	- If no more hops exist, or at target then delete self after time interval - Execute rest of actuators	- Go to next hop - Delete self
Sender		- Execute actuators	- Send info packets to receivers
Receiver	- Local agent types	- If local agent is info packet then increment info counter	- Increment info counter
DDoS packet	- Next hop to destination	- If no more hops exist, or at target then delete self after time interval - Execute rest of actuators	- Go to next hop - Delete self

Table 15: Metric types

Metric type	Measurment
Good info delivery	Overall number of info packets to reach good receivers during simulation
Bad info delivery	Overall number of info packets to reach bad receivers during simulation
Nodes destroyed	Total number of nodes no longer alive at end of simulation
SOMAS agents destroyed	Overall number of SOMAS agents destroyed during simulation
Malicious agents destroyed	Overall number of malicious agents destroyed during simulation
Agent activity	Aggregate change of agent count per location, summed over all locations
Feasibility	Whether simulation ran for total number of steps without running out of memory
Self organization	Amount of self organization evident in simulation history, where the history is composed of vectors designating a set of simulation statistics

Table 16: Network types

Type	Parameters
Flat	{nodes: 50, edges: 150, type: randomly interconnected}
Component	{{backbone | nodes: 5, edges: 10, type: fully interconnected}, {component | nodes: 2, edges: 1, type: wheel}, {component | nodes: 5, edges: 8, type: wheel}, {component | nodes: 8, edges: 14, type: wheel}, {component | nodes: 15, edges: 28, type: wheel}, {component | nodes: 23, edges: 44, type: wheel}, {outlier | nodes: 1, edges: 0}, {outlier | nodes: 1, edges: 0}}

5.6 Scenarios

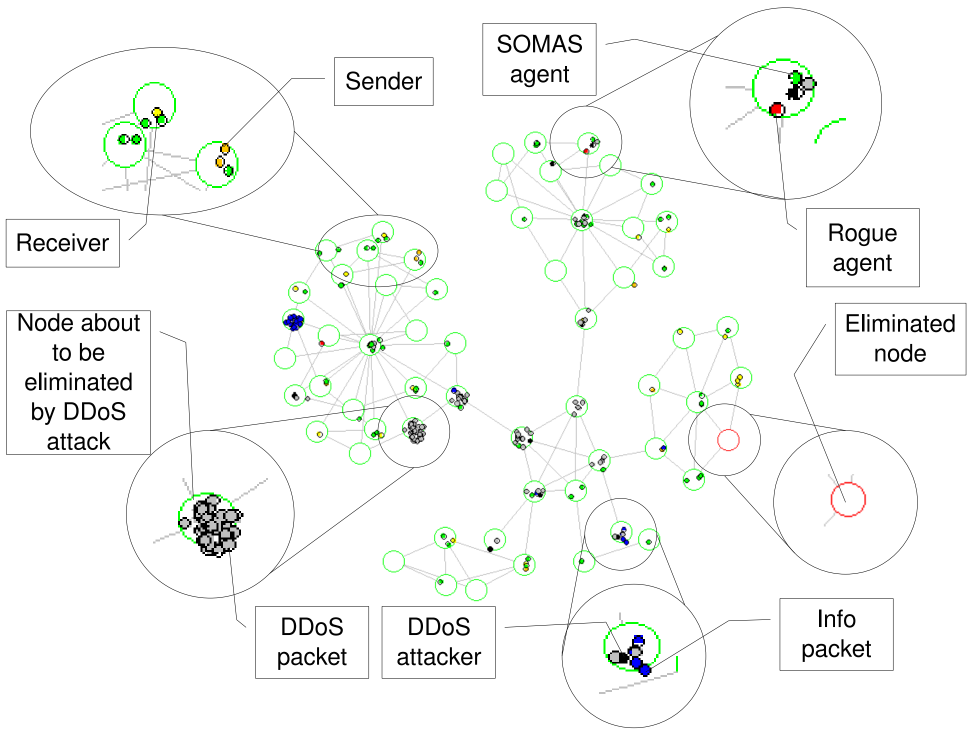
Table 17 gives an overview of the scenarios used in the chromosome evolution. Tables 14 and 15 list the types of agents and metrics used in the scenarios. For examples of the network layouts used, see the scenario figures. Figure 16 details a graph with a large number of network components and Figure 20 details a flat graph.

Table 17: Scenario types

Scenario	Agents	Metrics	Network type
All	- SOMAS	- Minimize activity - Maximize self organization - Maximize feasibility - Minimize number of dead nodes	
Vital Element		- Minimize marked nodes - Maximize gain in shortest path length - Maximize gain in number of disconnected subgraphs	- Flat
Intrusion Elimination	- Compromise - DDoS	- Minimize number of locations compromised	- Flat
Enemy Avoidance	- DDoS	- Minimize number of unsafe SOMAS agents - Maximize number of safe SOMAS agents	- Flat
DDoS	- DDoS - DDoS packets	- Minimize number of targets destroyed - Maximize number of malicious agents destroyed	- Flat
Competition	- Compromise - DDoS - DDoS packets	- Minimize number of SOMAS agents destroyed - Maximize number of malicious agents destroyed	- Flat
Information War	- Compromise - DDoS - DDoS packets - Sender - Receiver - Info packets	- Maximize amount of friendly infomation delivered - Minimize amount of enemy information delivered	- Component

- *Scenario Generalities:* For all scenarios except for Info Attack, every node, except for the outlier nodes, contains a single SOMAS agent at step 0. All flat graph network layouts are randomly constructed, given a set of parameters. The random construction takes place by randomly selecting and connecting two nodes per iteration, with the constraints that only the set number of edges can be created and the graph must be connected. The component graph network layouts all have the same topology. The parameters are detailed in Table 16. In the component based graph, all the components and outliers are connected to

Figure 16: Info War scenario



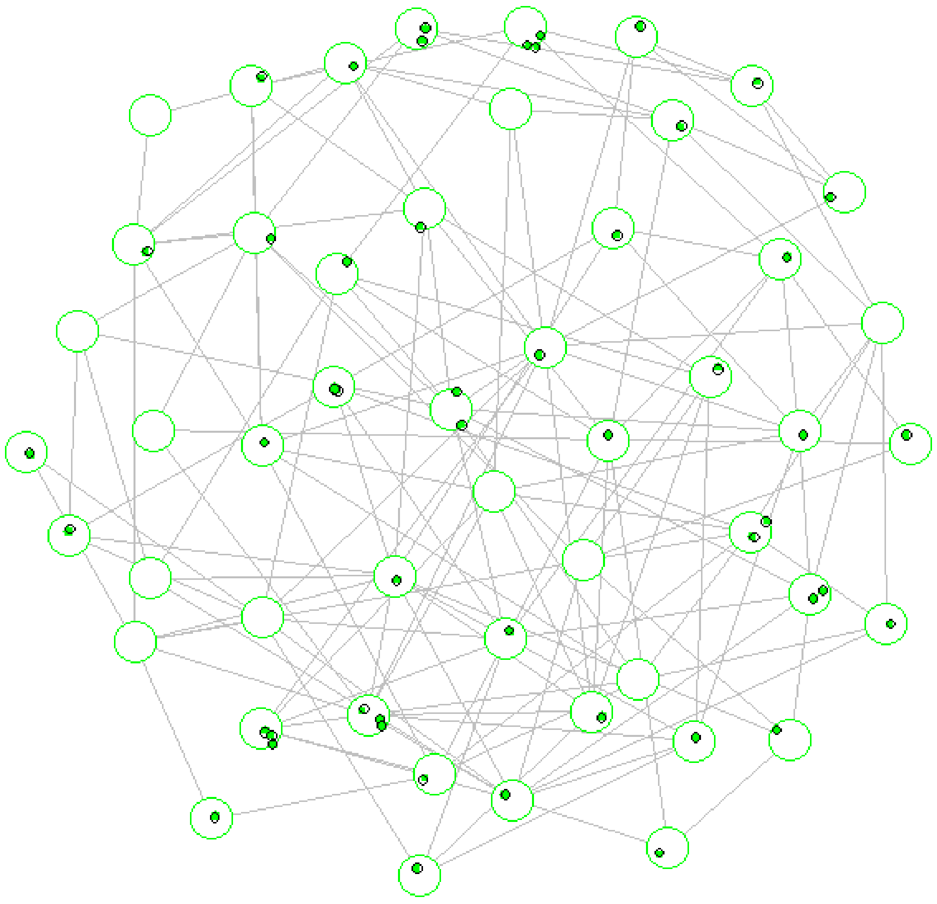
the backbone subgraph by a single link from one of the external nodes on the wheel.

Each of the scenario figures are an actual snapshot from the swarm visualizer. Figure 16 gives an overview of what each element in a scenario snapshot means.

1. *Vital Element Identification:* The vital elements are those elements in the network that cause the most amount of damage to the network with the removal of the least number of elements. Solving the problem optimally is an NP-Hard problem [19]. The SOMAS agents find a solution by marking and unmarking nodes as vital, and the nodes that are marked by the end of the simulation run are considered vital nodes.

2. *Intrusion Elimination:* A set of compromise agents randomly travel around the network compromising nodes and creating DDoS agents. The SOMAS agents attempt to uncompromise as many nodes as possible.

Figure 17: Vital Element scenario

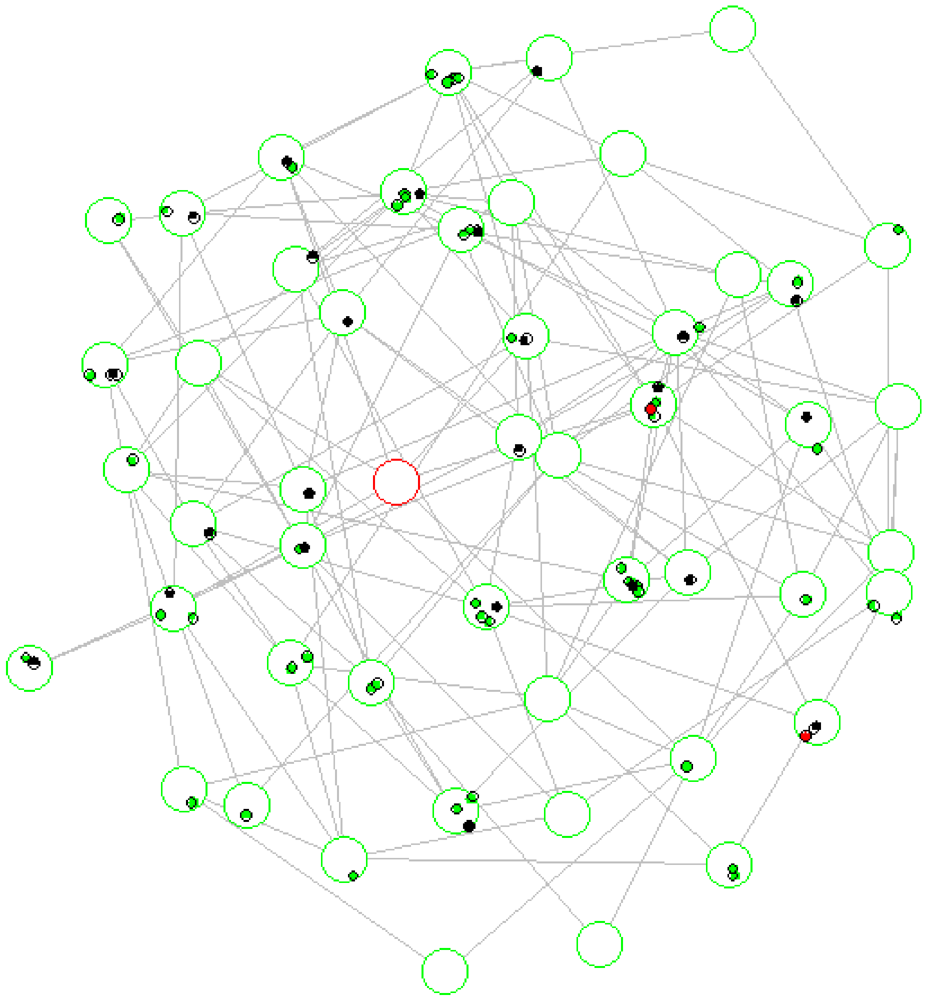


3. *DDoS:* In the network there are a set of unsafe nodes that the SOMAS agents attempt to avoid. The more SOMAS agents there are on the safe nodes the better, and this includes SOMAS agents creating more agents on the safe nodes and deleting agents from the unsafe nodes.

4. *DDoS:* In the network there are a set of target nodes that the DDoS agents attempt to destroy. The SOMAS agents use their capabilities to eliminate as many of the DDoS agents and packets as possible while also keeping all the targetted nodes alive.

5. *Competition:* Compromise agents and SOMAS agents are pitted against each other. The compromise agents attempt to destroy the network by compromising nodes and creating DDoS agents, which in turn mount a DDoS attack against

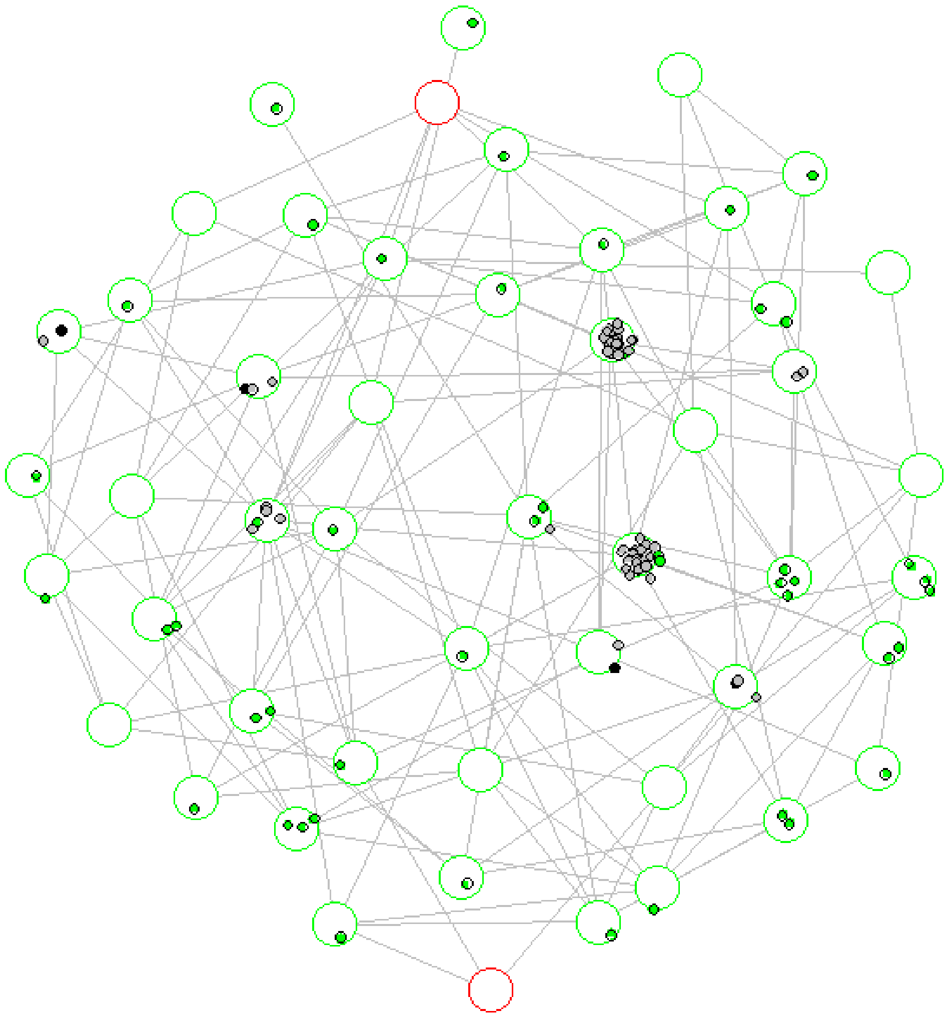
Figure 18: Intrusion Elimination scenario



every node on the network. Whenever a node is destroyed, all agents on the node are removed from the simulation. The SOMAS agents can delete agents they find and also destroy or bring back up network nodes. However, the SOMAS agents do not know which agents are friendly and which agents are malicious. The SOMAS agents have these same capabilities for the rest of the scenarios.

6. *Info War:* The Info War is the most complex scenario, and combines elements of most of the other scenarios. The network layout is composed of multiple subcomponents, instead of being flat. Within the network, there are a set

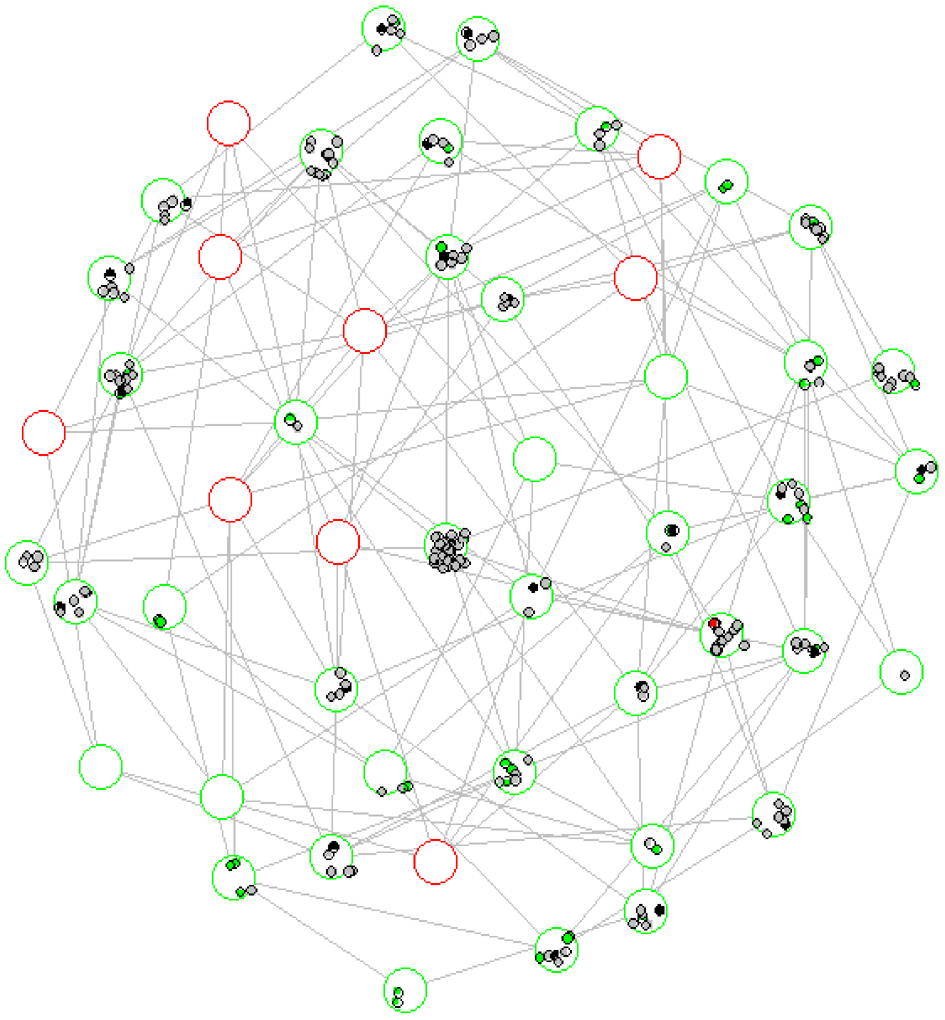
Figure 19: DDoS scenario



of sender and receiver nodes. The sender nodes attempt to continually send information packets to the receiver nodes.

There are also a number of compromise agents, who begin on nodes situated outside of the main network. They enter the network, compromised as many nodes as possible, and create as many DDoS agents as possible. Meanwhile, they also delete friendly information packets and steal the information and send it out of the network. The DDoS agents target all the receiver nodes in order to stop all friendly information traffic.

Figure 20: Competition scenario



Successfully completing this scenario requires that the SOMAS swarm use multiple behaviors from the previous scenarios. First, they must stop the receiver nodes from being deactivated by the DDoS agents. This is the same behavior that is necessary for the DDoS scenario. Second, the SOMAS agents need to stop the compromise agents from compromising the network nodes. The behavior is a combination of the behaviors from the Competition, DDoS, and Intrusion Elimination scenarios. Additionally, the SOMAS swarm must develop new behaviors to stop the compromise agents from stealing information.

5.7 Summary

This chapter describes how the effectiveness metrics, visualizations, statistical tests, behaviors, and scenarios are used to evaluate the significance of self organization and entangled hierarchies for creating effective SOMAS swarms. The effectiveness metrics and statistical tests are used to compare different SOMAS swarm types, and the behaviors and scenarios show the range of capabilities the SOMAS swarm possesses. The following chapter uses these test elements in the prescribed methodology to verify and validate, or invalidate the test objectives. This is accomplished both quantitatively with the tests, and qualitatively with the visualizations.

VI. SOMAS Simulation Analysis

6.1 Overview

Using the testing methods discussed in Chapter V, this chapter presents the results in order to determine whether specified hypotheses based on research objectives have been falsified or validated (since experimentation cannot verify hypotheses). Analysis of quantitative statistical results and visualizations is discussed in Section 6.2. Besides confirming or denying the hypotheses the results also describe a few other items of interest, which are covered in Section 6.3.

6.2 Hypotheses Validation

Table 18: Hypotheses results

Hypothesis	Outcome
- Self organization metric improves effectiveness of search	Partial success
- Visually identifiable self organization is evolved	**Success**
- An entangled hierarchy becomes more effective as the scenario becomes more complex	**Success**

Table 19: Legend of shorthand notation

Notation	Description
Non mod	Scenario is not modified beyond any changes mentioned in the caption
No evo	SOMAS agents do not evolve the chromosomes they carry
No c/d	SOMAS agents do not create or delete any of their own kind
Single	SOMAS agent chromosomes are evaluated on a single scenario
Multi	SOMAS agent chromosomes are evaluated on multiple scenarios
Dest	SOMAS agents can destroy network nodes
Non dest	SOMAS agents cannot destroy network nodes
w/ destruction	SOMAS agents can destroy network nodes
NV	SOMAS swarm does not have network node deactivation and activation actuators
NS	SOMAS swarm does not use online search
NE	SOMAS swarm does not evolve itself, but still uses online search
EV	SOMAS swarm uses evolutionary search

6.2.1 Self Organization Metric Improves Effectiveness of Search. The self organization metric is negatively correlated with a number of objectives:

- Maximizing Feasibility
- Minimizing good agent deletion
- Maximizing bad agent deletion
- Maximizing the delivery of information to good receivers

  It is positively correlated with:

- Minimizing agent activity
- Minimizing targetted location deactivation
- Maximizing uncompromised nodes
- Minimizing compromised nodes
- Minimizing delivery of information to bad receivers

The correlation of the feasibility metric with the other objectives is the complete opposite of the self organization metric. This shows that the self organization metric primarily measures the inverse of feasibility. However, this does not mean that it is equivalent to the inverse of feasibility, which will be explained in Section 6.3.5.

The reason for this inverse relationship is because with a small number of agents the simulation history is fairly undifferentiated and long. Thus, the clustering algorithm used to extract the causal pasts (see Section 2.5.4) creates a minimal number of clusters and there is not very much predictive information that can be potentially extracted. However, with a large amount of agents, the simulation history is both irregular and is likely short due to memory overflow. These two influences cause the clustering algorithm to create a larger number of clusters, increasing the potential amount of predictive information that can be extracted.

Despite the lack of success optimizing the search using the self organization metric, self organization is seen in another aspect of the search. The SOMAS offline search demonstrates a phenomena known as self organized criticality [17]. While interesting for the possible uses of such a phenomena, it also appears to pull the search out of a local optima. These issues are discussed more completely in Section 6.3.5.

Therefore, the first hypothesis is **partially successful.**

6.2.2 Visually Identifiable Self Organization is Evolved. The visualization produces a simple network topology showing the types of agents at each node. Thus, the visualization can only show self organized behavior in terms of agent creation, deletion, fitness values, and movement. The SOMAS framework evolves various emergent behaviors, but they vary in the amount of self organization also exhibited.

The simplest strategy that emerges in the Information War scenario is that of doing nothing. Amongst the SOMAS swarms that can deactivate network nodes, a common strategy is to deactivate nodes until an objective is met, whether that means deleting all enemy agents or stopping a DDoS attack. This deactivation is carried out either by directly deactivating nodes, or by creating so many agents on a node that the node automatically deactivates. However, this behavior does not show organization amongst the swarm agents.

A less destructive behavior in the Information War Scenario is to restart the DDoS'd nodes. The enemy DDoS agents have a specific order of targets they are meant to deactivate, and will not proceed to the next target as long as any previous targets are active. By reactivating the DDoS'd nodes, the DDoS agents are kept from deactivating the other receiver nodes.

Besides these simplistic emergent behaviors, there are also emergent behaviors that exhibit greater organization in the swarm. In the Competition scenario there is a strategy that produces swarms that move as groups around the network, deactivating nodes. The swarm is able to halt the compromising of network nodes and consequent DDoS attack against the network.

There are solutions in the Information War scenario where the swarm will activate a network destruction and creation process when under attack by DDoS agents.

The agents will start reproducing very rapidly and DoSing nodes themselves, as well as restarting deactivated nodes. Once this behavior begins in one component of the network, the behavior rapidly spreads itself to the other components. Unfortunately, the behavior eventually leads to receiver nodes being destroyed and restarted, which limits the good information flow. However, the behavior also can delete sender nodes, and this is beneficial insofar as the amount of information that can be stolen is limited.

The reason why this behavior evolves is because it both restarts nodes that have been DDoS'd, thus ensuring enemy DDoS agents remain fixated on one target, and deactivate the bottlenecks in the network, which keeps the malicious agents from compromising network nodes in the network components and creating new DDoS agents. Thus, the SOMAS agents are creating a decoy for the DDoS agents.

These behaviors verify the second hypothesis, it is consequently **successful**. This is a verification instead of only a validation because the SOMAS framework has demonstrated this capability.

6.2.3 Entangled Hierarchy More Effective as Scenario Becomes More Complex. The statistical tests, which are found in Appendix B, demonstrate that as the scenario becomes more complex, swarms capable of more complex and influential entangled hierarchies become more effective than less capable swarms. As described in Table 12 in Section 5.2 the the scenarios are categorised according to the complexity of the network layout, and whether the environments and agents are static/dynamic. The figures in appendix B give a visual representation of how the categories affect the search landscapes. As shown, the online search swarms produce a wider variety of objective values in the offline search space. This suggests these swarms are able to acquire a greater number of building blocks and construct better solutions.

The results plots in Appendix B show that in almost all network scenarios, swarms with online search (EV and NE swarms) decisively outperform swarms that lack online search. Not only are online search swarms more capable than non searching swarms in more complex scenarios, the searching swarms are more capable almost across all scenarios.

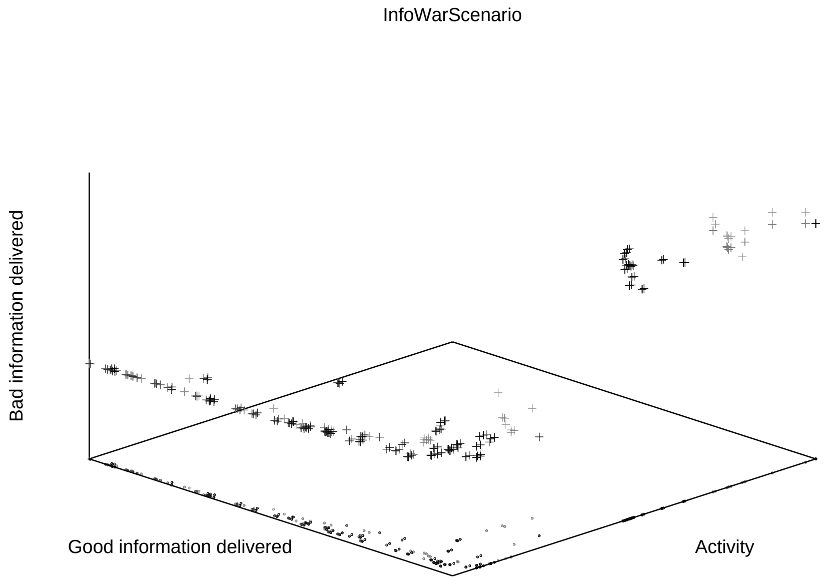
This successfully validates the third hypothesis in Table 18, so it is **successful.**

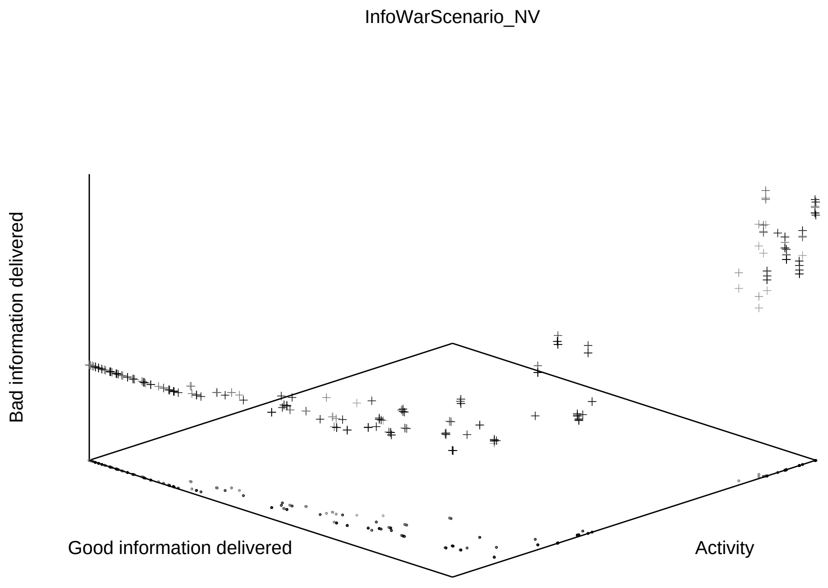
While the use of online search explains why the EV and NE swarms do better than the NS swarm, it does not explain why the EV swarm outperforms the NE swarm in only certain scenarios. Contrary to expectations, the increase in the EV swarm's effectiveness does not strictly follow the increase in scenario complexity, though such a trend is evident.

The distinguishing feature is that the scenarios where the EV swarm is more effective require a greater specificity of behavior than the scenarios where the NE swarm is more effective. For instance, even though the complexity of the Competition scenario is greater than the complexity of the DDoS scenario, the actual activity in the Competition scenario is much more homogeneous. In the Competition scenario it isn't important to perform specific actions in specific portions of the network, since the elimination of enemy agents is worth the same amount regardless of network location. Thus, the same swarm behavior is generally effective regardless of swarm location.

In the DDoS scenario the swarm has to protect specific targets, so the swarm needs to use different behaviors for different parts of the network. Thus, the swarm in this scenario needs to be capable of greater specificity in its behavior. The same pattern can be seen with the Information War scenario. When the swarm has node deactivation and activation actuators the EV swarm is more effective. This is because the swarm defends the network better if it focusses on reactivating the nodes targetted by DDoS agents, as explained in the previous Section 6.2.2. On the other hand, without these actuators, the swarm cannot be as specific in its behavior, so the greater capability for specificity is no longer needed.

Figure 21: Pareto fronts comparing evolution with and without network destruction, part 1

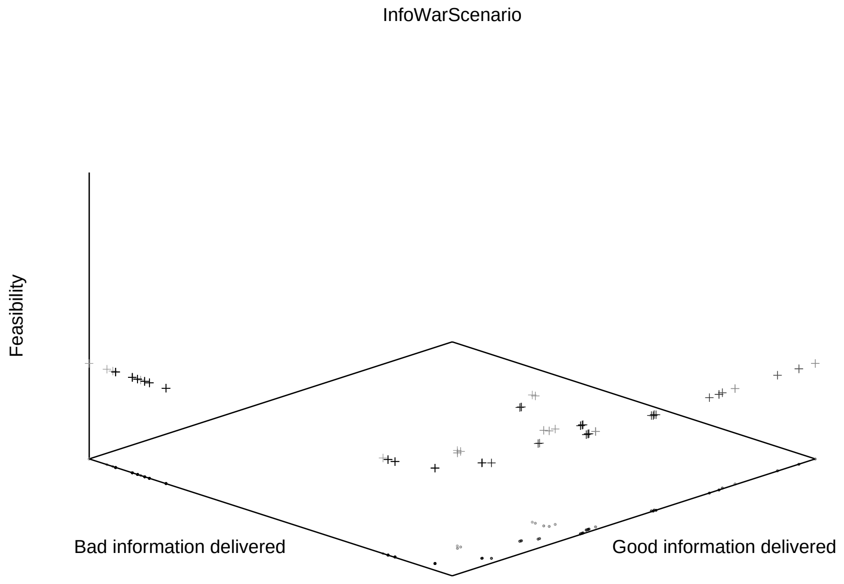


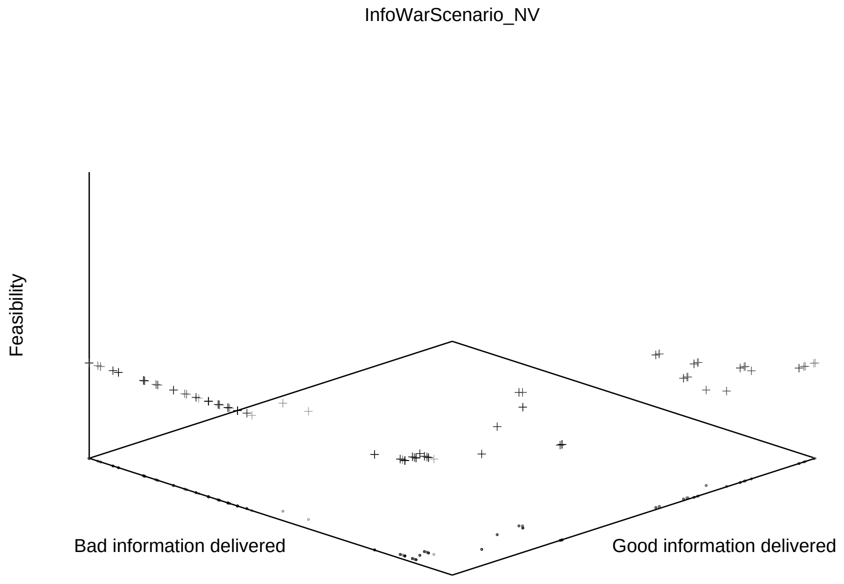


6.3 Secondary Testing and Analysis

Besides the hypotheses for testing the research objectives, a number of other tests and analysis are carried out for problem exploration. As noted with swarm

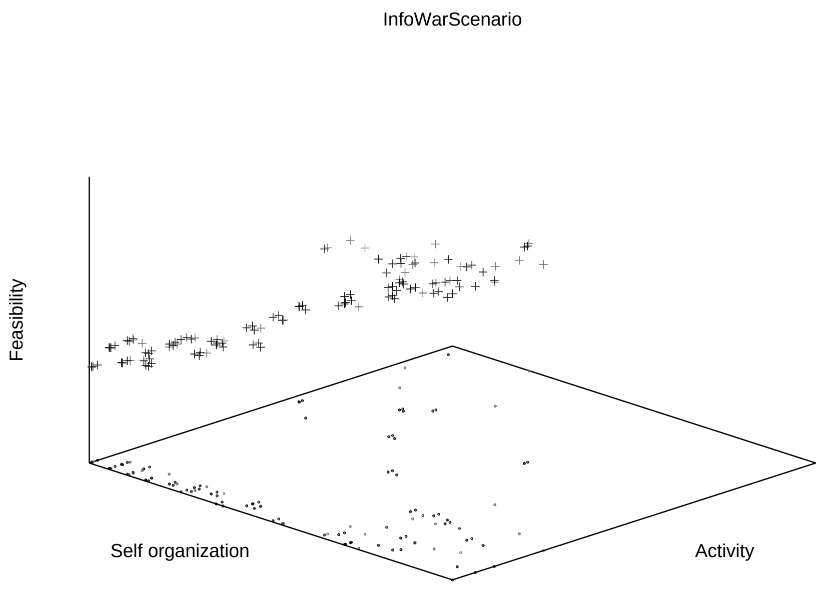
Figure 22: Pareto fronts comparing evolution with and without network destruction, part 2

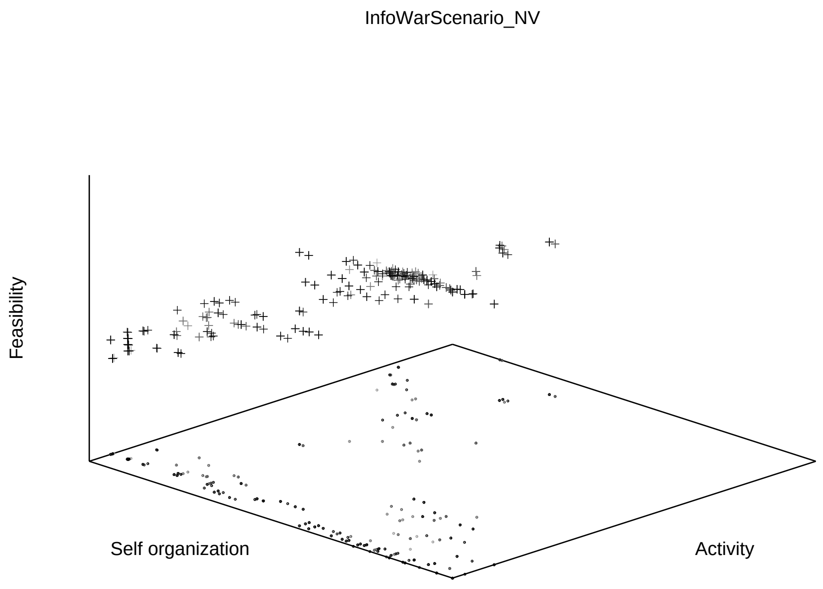




behavior effectiveness, the ability of the swarm to destroy the network significantly changes whether swarm evolution is preferable or not. More tests are carried out

Figure 23: Pareto fronts comparing evolution with and without network destruction, part 3





between swarms capable of destruction and swarms incapable of destruction to more precisely determine there respective effectiveness.

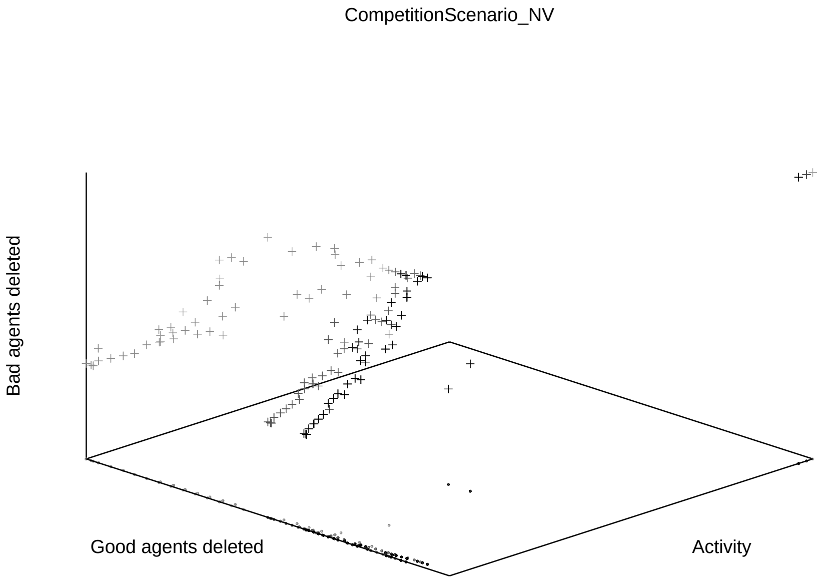
During the discussions in the previous chapters, it is mentioned that the swarm behaviors can be composed of sub behaviors. This claim is tested by evaluating swarm chromosomes on multiple scenarios during the course of the search to see whether simpler scenarios can be combined to create more effective complex behavior.

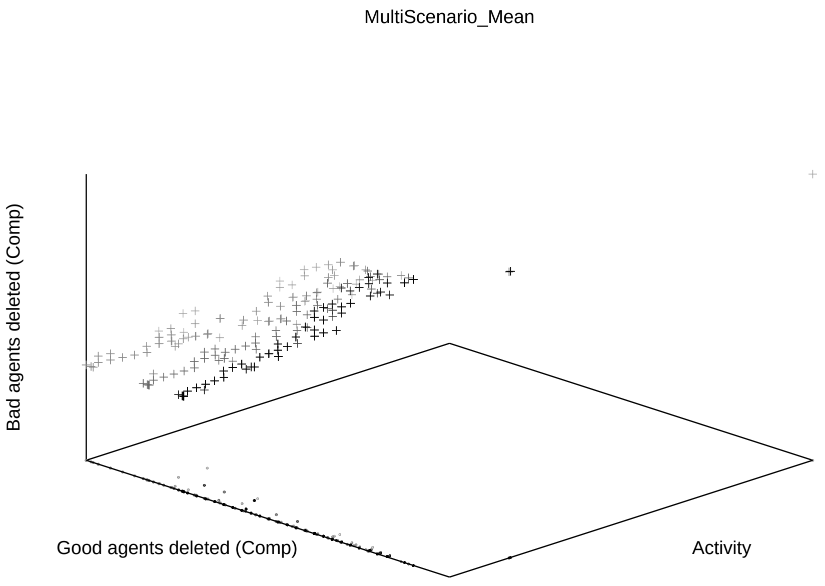
Further analysis is carried out on the accomplished tests to look for other characteristics besides those originally being tested. Interesting results are found, such as evidence for the efficacy of differential evolution, objectives from disparate scenarios working together to overcome a royal road, and the existence of self organized criticality during the search process.

6.3.1 Effectiveness of Network Destruction. Generally, the swarms with node deactivation and activation actuators decisively outperforms the swarms that do not have these actuators, as shown in Appendix C. However, the exception is in the case of the two scenarios where specificity of action is not significantly required (Competition and Intrusion Elimination scenarios) and with the swarm less capable of specific action (non searching swarm). Thus, the results corroborate with the conjecture from the previous experiments that the online search swarms are so effective because they are capable of more specific behavior.

6.3.2 Multi Scenario Evolution. For the DDoS and Information War scenarios using multiple scenarios for evolving a chromosome works much better than using a single scenario. For others, the results are ambiguous. These results are shown in Appendix D. The cause is most likely the dependency between scenarios. The set of behaviors necessary for success in the Competition and Intrusion Elimination scenarios is a subset of the behaviors for the Info War and DDoS scenarios, as explained in Section 6. Thus, all progress in the former scenarios helps the latter, but progress in the latter does not necessarily help the latter. In fact, the behaviors may become too complex to benefit the simpler scenarios. Figures 24, 25, 26, and 27 compares the exploration of single and multi-scenario evolution. For the Information War plot, the

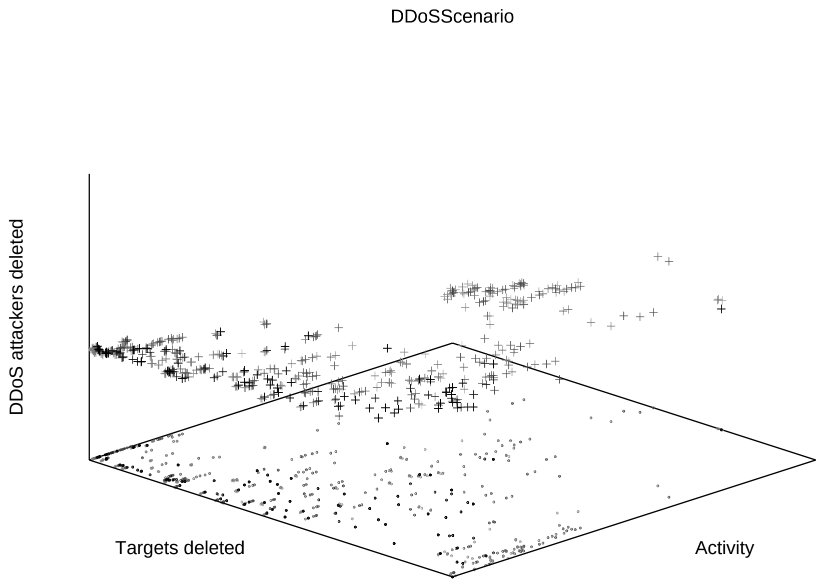
Figure 24: Pareto fronts comparing explorativeness of single and multi-scenario evolution, part 1

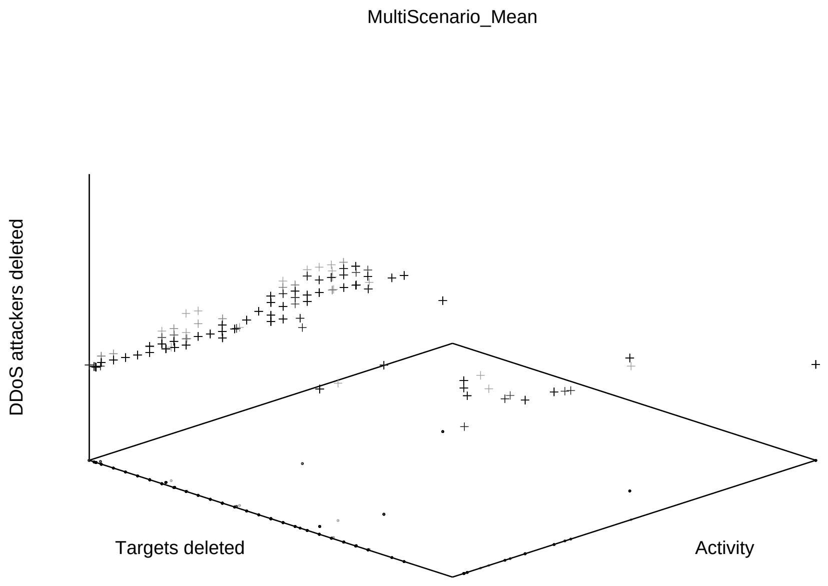




multi-scenario results converge clearly better than those of the standard scenario. In the others, it is much harder to judge, even in the case of the DDoS scenario.

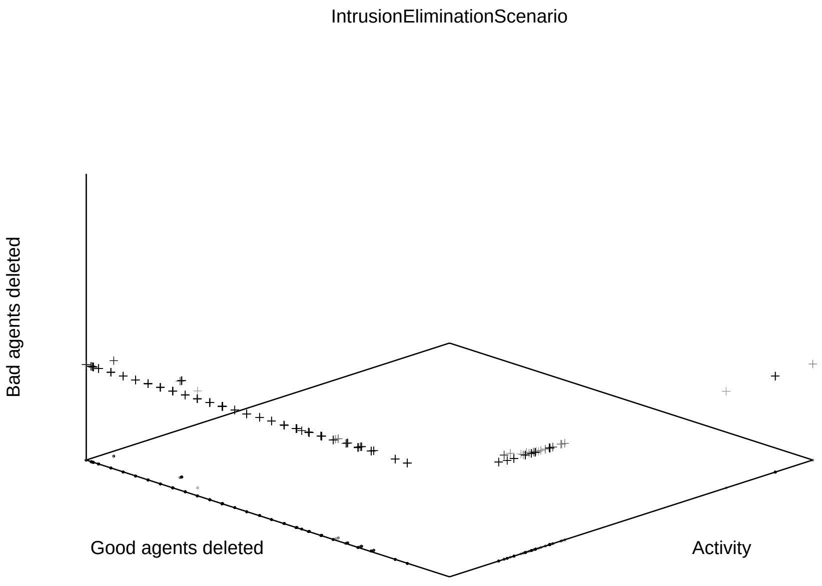
Figure 25: Pareto fronts comparing explorativeness of single and multi-scenario evolution, part 2

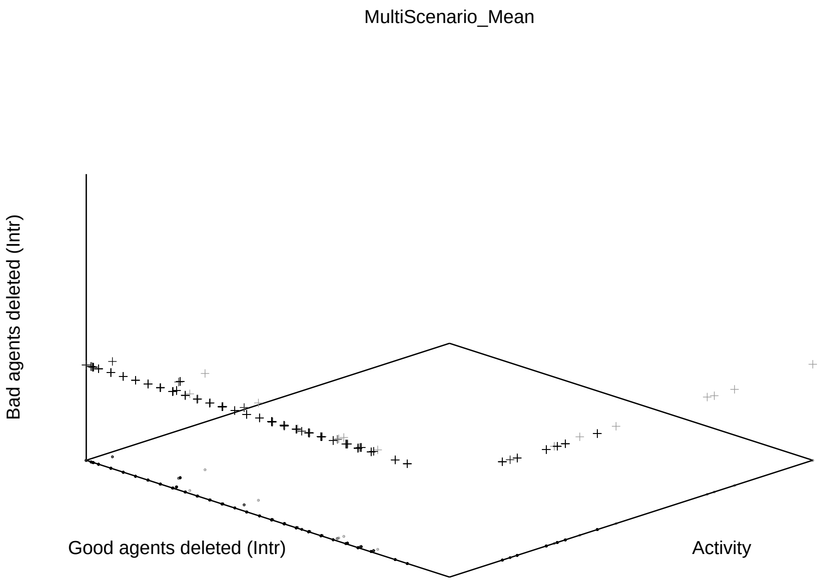




6.3.3 Evidence of Differential Evolution Usefulness. In a couple of the scenarios, there seem to be two primary strategies that shape the direction of evolutions, see Figure 28. Even though in MOPs there is no guarantee a pattern in the geno-

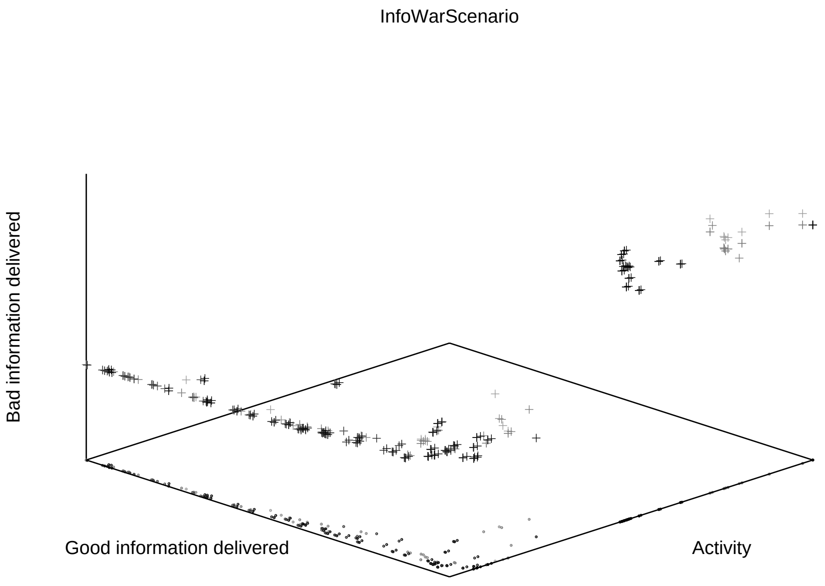
Figure 26: Pareto fronts comparing explorativeness of single and multi-scenario evolution, part 3

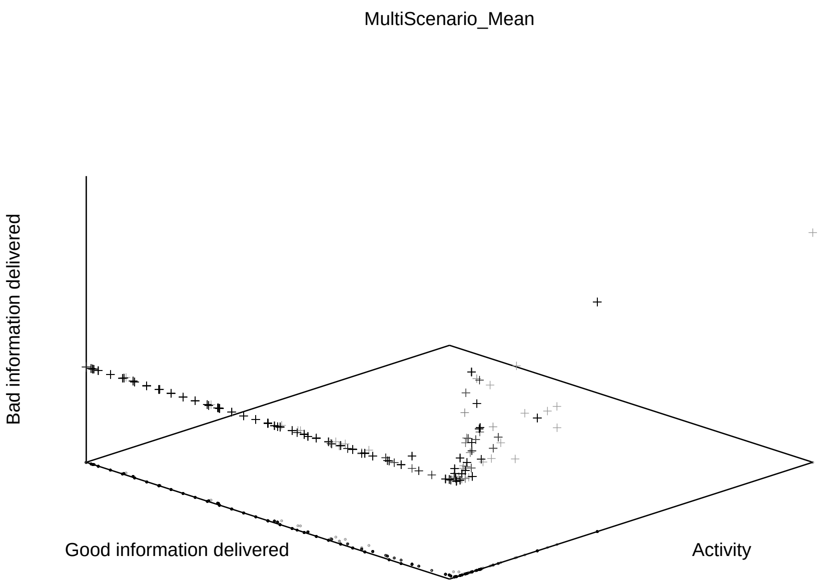




type implies a pattern in the phenotype, such consistency is suggestive that there is. If this is the case, then these scenarios can benefit from differential evolution using real values instead of bit strings. Differential evolution is a combination of gradient

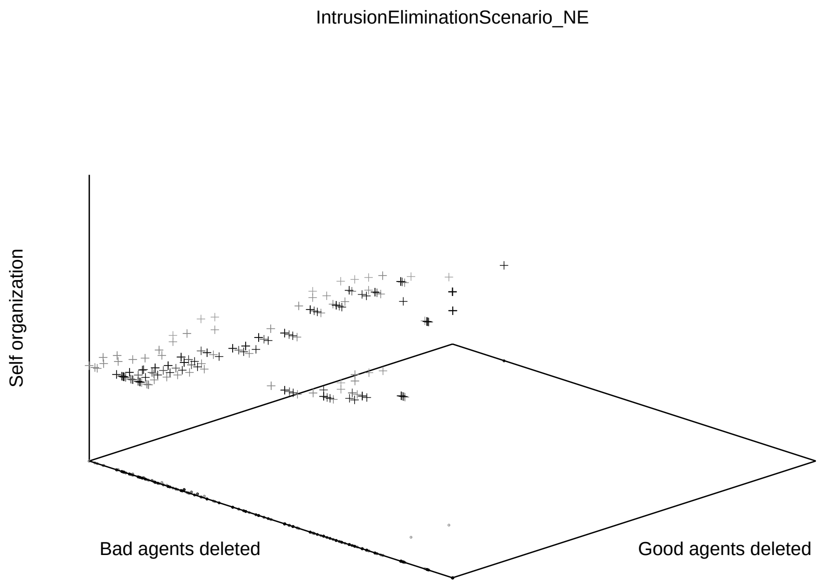
Figure 27: Pareto fronts comparing explorativeness of single and multi-scenario evolution, part 4

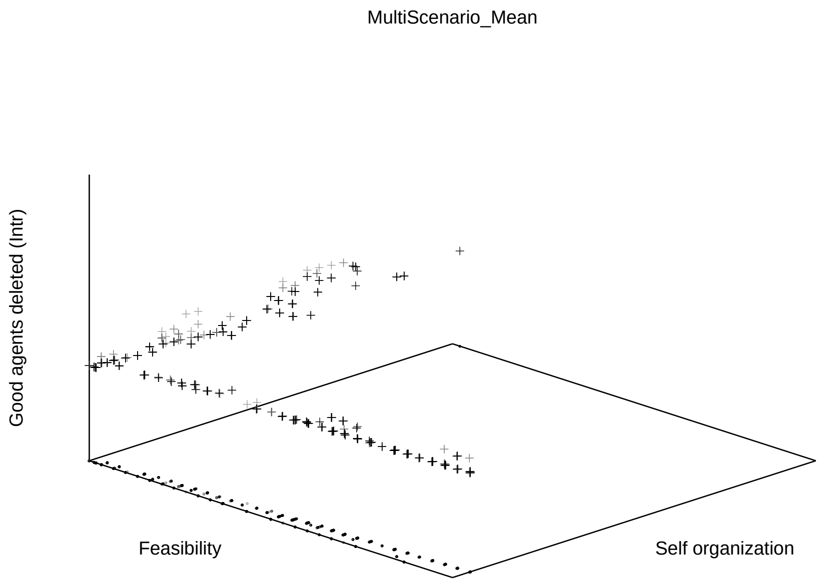




descent and genetic algorithms, as described in [125]. By finding the rate of change between the objectives of chromosomes, differential evolution may be able to adapt to the patterns in the plots and find the optimal points much more quickly.

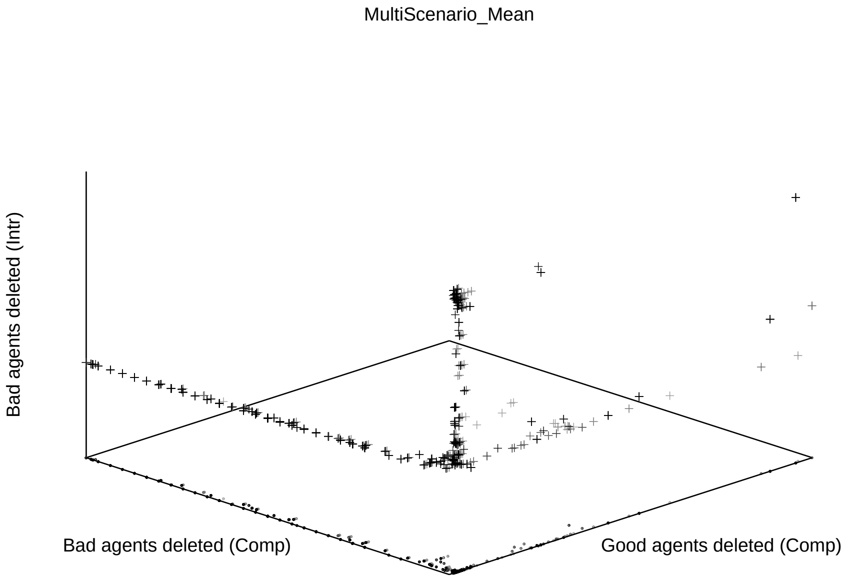
Figure 28: Two strategies for intrusion elimination

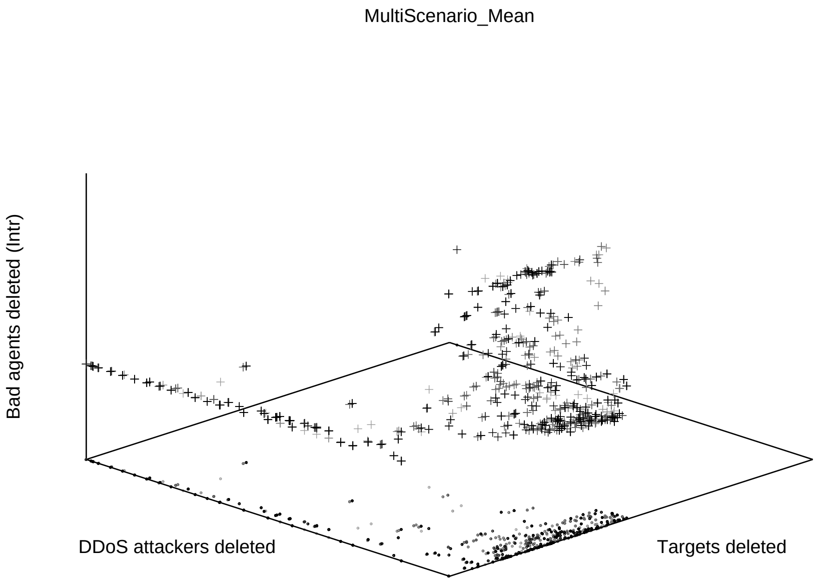




Nowak researched the use of differential evolution for evolving self organized swarm behaviors, and demonstrated its superiority over neural networks for his problem domain [89]. Given the similarity between the his work and this thesis it is possible differential evolution can follow the linear patterns in the pareto plots and increase the search's convergence rate, while retaining the same level of exploration.

Figure 29: Independent and interacting objectives





6.3.4 Objective Dependencies In Multi-Scenario Evolution. While it is not surprising when objectives from different scenarios are independent, it is strange when the addition of a foreign objective to a PF_{known} plot results in objectives from the same scenario becoming independent as well, especially when ostensibly the foreign objective should be the same as one of the local objectives. Yet, this is shown in the first plot of Figure 29. The literal meaning is that improvement in the foreign objective, of deleting more bad agents, is uncorrelated with improvement in either of the other two objectives. The main difference between the two scenarios from which the objectives are derived is that in the Competition scenario the DDoS agents that the rogue agents create can destroy the network, and consequently destroy the SOMAS agents, whereas the SOMAS agents are under no threat in the Intrusion Elimination scenario. So, the independence may be due to SOMAS agents not needing to protect themselves.

The second plot shows that foreign objectives are not always independent of local objectives. There is a clear interaction between the number of targets deleted by DDoS agents and the number of intruders deleted. This makes sense, because in both cases being able to delete DDoS agents is beneficial. However, it is again surprising that the DDoS attackers deleted is fairly independent of bad agent deletions.

The discrepancy is possible a result of the difference in ideal strategy between the scenarios. In the Intrusion Elimination scenario a node is cleaned from intruders if it is destroyed, meaning a very simple and effective strategy is to merely destroy all the nodes in the network. Consequently, most chromosomes probably localize on variants of this strategy, resulting in different numbers of target nodes being destroyed. Such a strategy does not work well in the DDoS scenario since the number of targets destroyed would be maximized. Thus, the SOMAS agents need a more sophisticated approach to rid the network of DDoS attackers, and the strategies for malicious agent deletion between the two scenarios do not correlate. Visualizations of behaviors from these scenarios show validity for this hypothesis.

Figure 30: Objective attractor

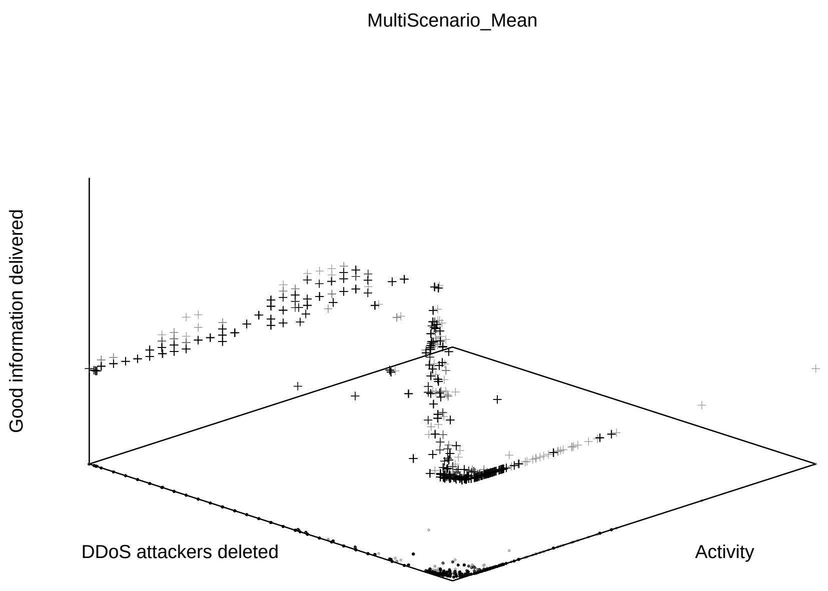
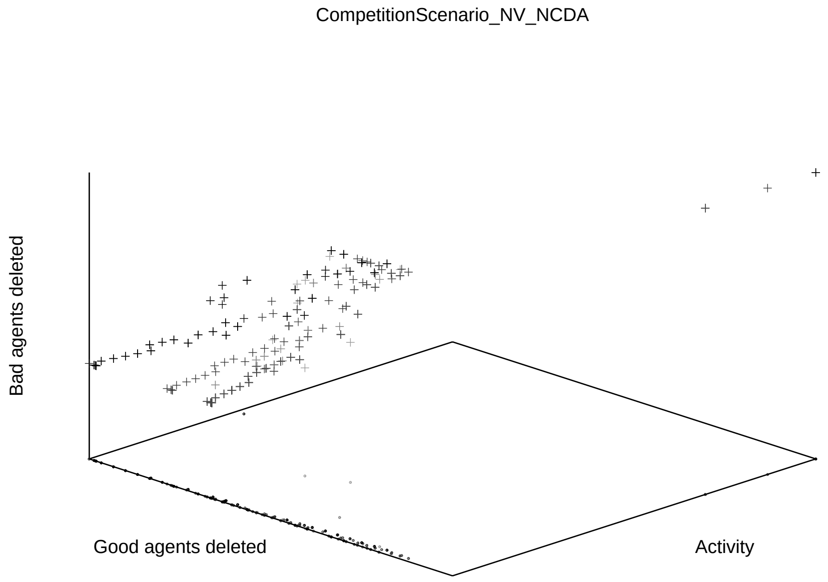


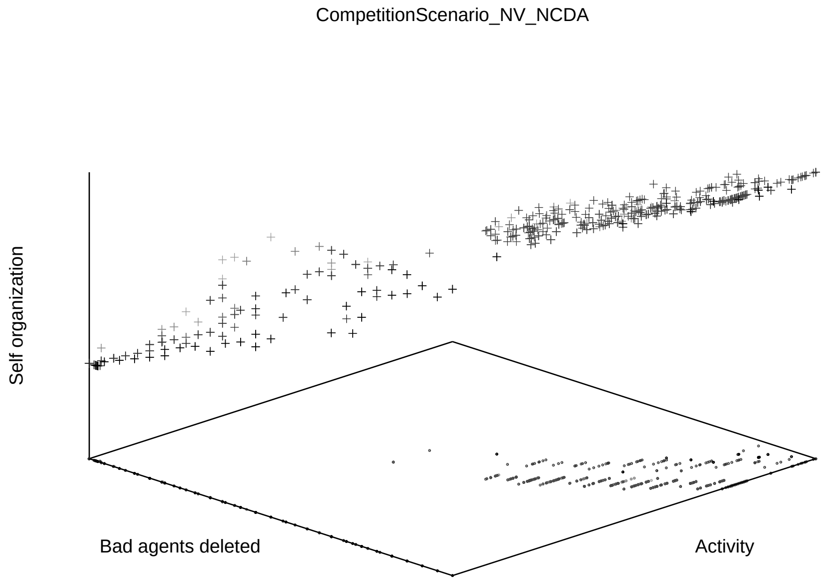
Figure 30 shows two interesting aspects of multi-scenario evolution. First, something of a royal road exists in combination of the Info War and DDoS objectives; they are inversely correlated until a drop off point. If the activity objective were not included, then the tradeoff between the two would hamper evolving chromosomes for either scenario. But, the addition of the activity objective pulls the path over the royal road's cliff, ending in an attractor right around the optimal solution. Thus, synergy is created by grouping objectives from different scenarios.

The second interesting characteristic is the attractor. The previous plot of the independent objectives in figure 29 and the plot of the royal road show attractors in the solution space. These do not seem to be prevalent in the PF_{known} plots for single scenario evolution. Perhaps this is due to the commonalities between the ideal behaviors in the different scenarios. These commonalities will amplify the success of certain chromosomes since their success translates to most of their objectives.

6.3.5 Self Organized Criticality. Throughout many of the scenarios, the PF_{known} plot shows an intriguing gap in patterns of dominant chromosomes. Chromo-

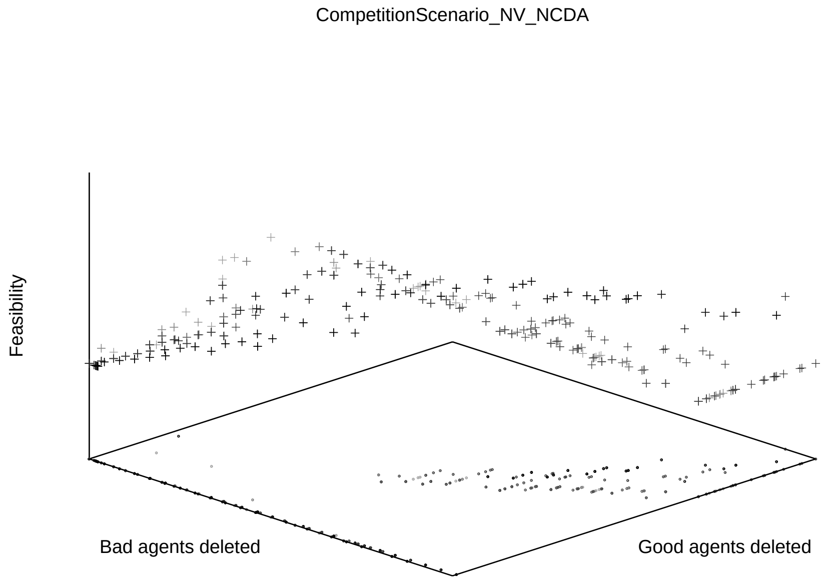
Figure 31: Pareto fronts showing objective relationships for self organized criticality, part 1

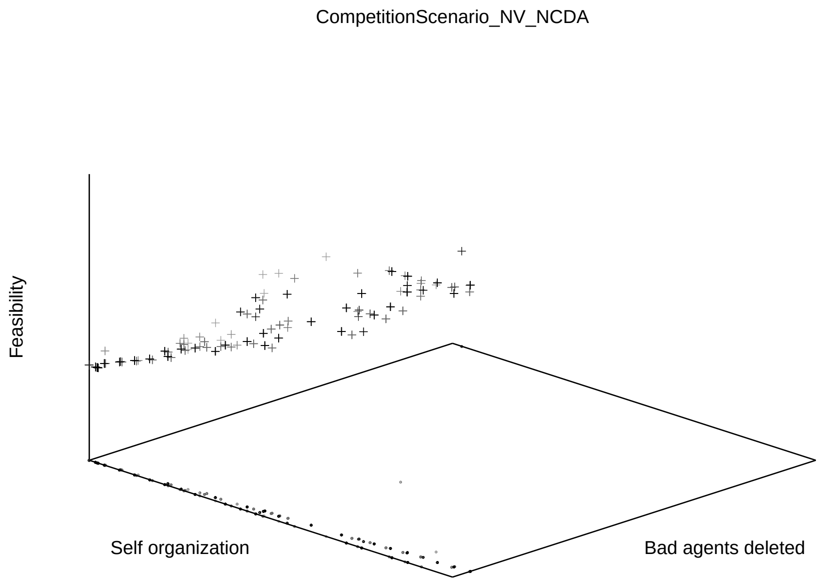




some evaluation produces fairly tight variances along a path up to a certain midway point, then the variance becomes large very rapidly and certain sections along the path do not contain any values. By comparing the different objective plots, it is clear

Figure 32: Pareto fronts showing objective relationships for self organized criticality, part 2





that this relationship holds between the feasibility, self organization, and bad agent deletion. This pattern does not also hold between activity and good agent deletion because the number of good agents is static if they are not deleted while the number of bad agents increases if they are not deleted.

Figure 33 shows that the gap disappears when non creating agents are compared to creating agents, but it shows up again when both kinds of agents are creative. The question is, why is there a gap at the midway point? Due to the chaos created by many agents, it might be assumed that the greatest variance would be found at the path end with the most agent activity.

The discrepancy is explained by the concept of self organized criticality [17]. In self organized systems, self organization reaches its highest level at a point right before chaos, where most of the system's energy levels are filled. Thus, a disturbance in the system results in an avalanche of energy dissipation.

Since the system is so close to chaos, yet is very well ordered at the same time, there is great potential for a variety of behaviors to emerge. However, when the system is not very self organized, due to singificant chaos or simple behavior, there are not many new behaviors that can develop. Consequently, the greatest range of behavior, and thus the greatest range of objective values, will be found around the point of self organized criticality. Self organized criticality can even lead to new regions of the solution space that would not otherwise have been reached, as in Figure 34.

There are a couple uses for this section of self organization. First, it is useful for finding effective behavior. Figure 34 shows how criticality can benefit the EA's search. Before the point of criticality, the path of the objective values shows an inverse correlation. Evidently, there is an effective strategy that is restricting the search. The point of criticality allows the search to find a new strategy that introduces a correlation between the objectives, leading to an attractor much closer to the optimal value.

Second, it can be used as a feedback mechanism for the self organization metric. The plots show that the metric is not accurate, since the optimal level of self organization is not found at the point of criticality. During the development of a self organization metric, its effectiveness can be measured by determining how closely it identifies the criticality point.

6.4 Summary

The test results for hypotheses to validate the research objectives are analyzed to see whether the hypotheses are valid. For the first hypothesis, that the self organization metric optimizes the search, it is shown that the self organization metric does not succeed in totally optimizing the search. It correlates positively with some objectives and negatively with other objectives. At the same time, swarms are evolved that do exhibit self organized behavior when visualized.

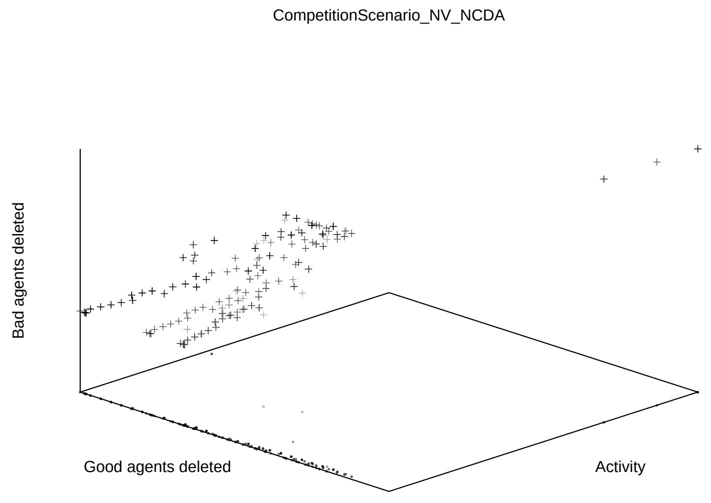
The entangled hierarchy hypothesis is successfully validated. The test results show that as the scenarios become more complex, the swarms with entangled hierarchies are more effective than those that do not have entangled hierarchies, or have entangled hierarchies to a lesser extent. Additionally, with more entangled hierarchies the swarms are capable of more specific behaviors.

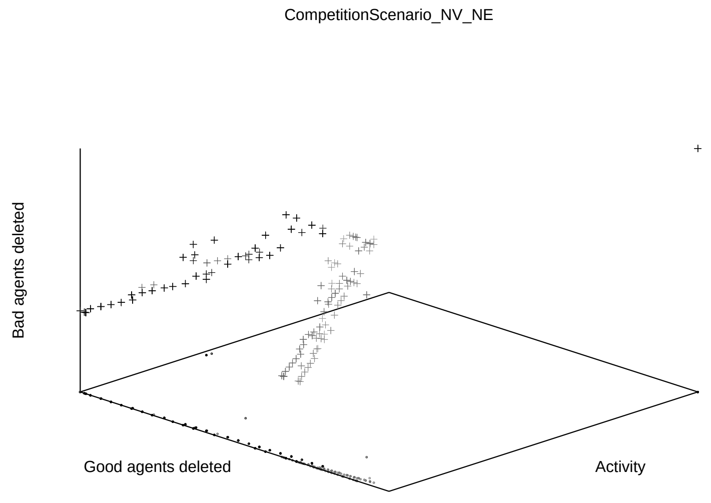
Based on the first set of testing and analysis, new tests are run to investigate features of the SOMAS capabilities, and more analysis is done on the tests to identify other characteristics. In the testing of entangled hierarchies, it is noticed that the swarm's ability to deactivate and activate the network significantly affects whether an entangled hierarchy is a benefit. Further tests show that swarms that use online evolution without network deactivation and activation actuators fare much worse than swarms that can destroy the network. Again, this result is linked to the greater capability for specific behavior exhibited by the swarms with more of an entangled hierarchy.

Another aspect of SOMAS that is tested is whether using sub-behaviors in the objective space can evolve more capable behaviors. However, this experiment does not show success in evolving the behaviors, though some of the sub-behaviors are evolved more effectively. The plots do show objectives from different behaviors working together to improve the search, so there is evidence that the technique may work. There is just no conclusive evidence that the technique is universally effective.

When the results are analyzed for other characteristics, a few new findings are observed. There is evidence that differential evolution may work well in the SOMAS problem domain. Even though the self organization metric did not prove especially useful, the Pareto plots of the search show that self organized criticality plays an important role in the optimization of certain objectives. All of these results pave the way for important conclusions and extensive future research. The conclusions and future research are covered in the next chapter.

Figure 33: Evidence that self organized criticality depends on agent creation





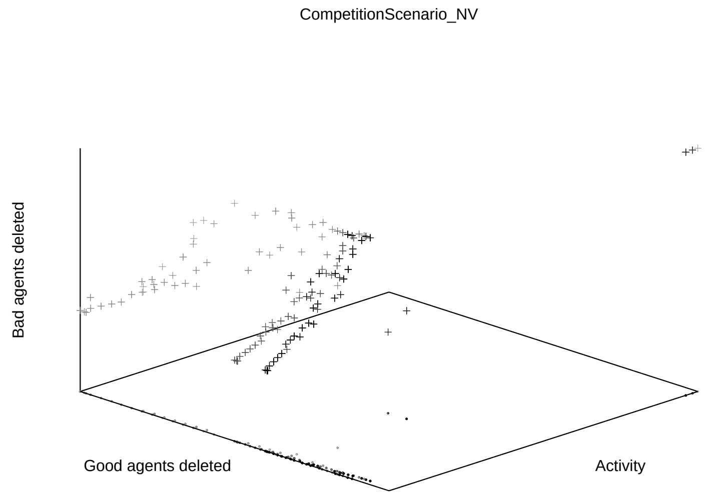
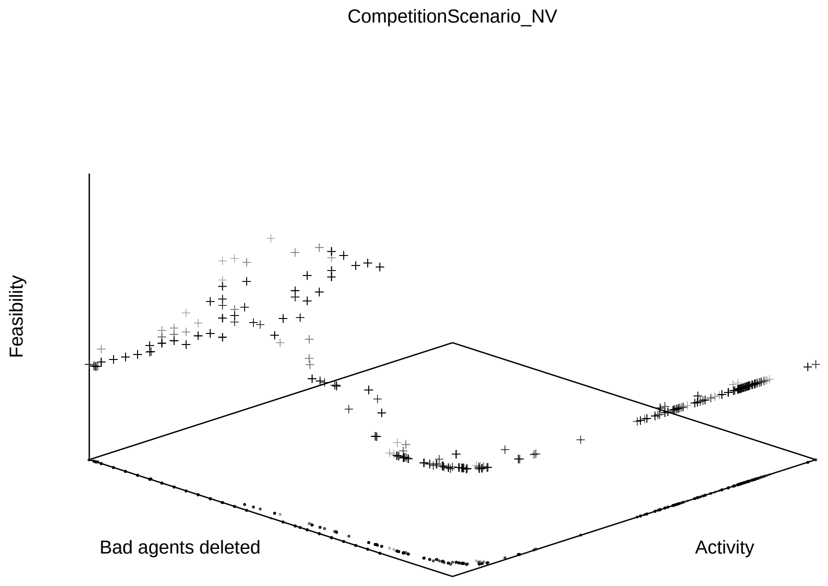


Figure 34: Self organized criticality leading to optimal



VII. Conclusions and Future Work

7.1 Summary of Research

Overall, the objectives in Chapter I have been achieved. Based on the background material in Chapter II, the SOMAS architecture is successfully created in Chapters III and IV. The self organization metric correlates with certain behavioral objectives and guides the search towards a region of self organized criticality. However, the metric does not serve to totally optimize the objective attainment since it is inversely correlated with some SOMAS behavioral objectives. Finally, the effectiveness of a self evolving swarm on a more complex scenario has been satisfied. These results are shown with the tests and results analysis in Chapters V and VI.

The experiments in Chapter VI show self organization does play an significant role in the search process for effective SOMAS, see Figure 5, through self organized criticality. While the extremes of the search generally have small variance, the midpoints of certain scenarios contain potentially chaotic regions that produce both good and bad chromosomes. Their high level of exploration can also lead to areas of the search space that may not have been found otherwise. These chaotic regions are explained by the theory of self organized criticality. However, the metric used to measure self organization should be developed further.

While the experimentation with self organization is not totally conclusive, the experiments with entangled hierarchies in Chapter VI, see Figure 10, show definitive results. When swarms can evolve online they are much more effective in complex scenarios than swarms that cannot evolve, as is hypothesized. Since online evolution is adaptation through the use of entangled hierarchies, the success of these swarms is an indication that entangled hierarchies provide the needed flexibility for network security management.

The following lists the objectives from Section 1.2 and whether they have been successfully accomplished. Multi-objective non-parametric testing and analysis statistically validates objective hypotheses for objectives 2 and 3. Pareto plots show the search history for feature identification and correlation, swarm simulations visualize behaviors for observational verification of self organization in Sections 6.2 and 6.3.

Objective 1. *Create a robust simulation framework for evolving self organizing multi-agent systems (SOMAS)*

Success: Using the MASON and PISA packages, and customized coding, SOMAS swarms are effectively evolved and evaluated.

Objective 2. *Evaluate effectiveness of self organization for accomplishing objectives*

Partial success: The self organization metric does not result in total optimization for all behavioral objectives. However, the search exhibits useful self organized criticality. Chromosomes are evolved that exhibit clearly self organized behavior when visualized. The latter two pieces of evidence show that self organization has utility for SOMAS.

Objective 3. *Evaluate effectiveness of entangled hierarchies for accomplishing objectives*

Success: Experimental results show that as the scenarios become more complex, entangled hierarchies become more effective.

However, these particular behaviors only demonstrate SOMAS feasibility and are not totally comprehensive of the system's capability. SOMAS can be applied to a wide variety of other network security problems, such as intrusion detection, network defense, hiding high value targets, etc. as well as issues not only restricted to network defense. Some of these areas of exploration are described in the following section, along with other methods of improving SOMAS.

7.2 Future Research

There are numerous potential research projects motivated by this investigation. They range from the theoretical to particular applications.

7.2.1 SOMAS Improvements.

1. The software needs to be written with better software engineering practices. It needs to be better organized and incorporate the formal model checking algorithm.
2. The offline evolutionary algorithm can adapt itself to the landscape by encoding MOEA paramaters in the agent chromosome, similar to the technique used in adaptive evolutionary strategies. However, if the landscape is dynamic then the adaptive algorithm may not be able to adapt to the dynamics, especially if they are noncyclic, and thus may be worse than just Monte Carlo sampling.
3. Since artificial intelligence has not generated any kind of mechanism resembling human intelligence, it is crucial to incorporate humans into the SOMAS production loop. As an analogy, computer systems are autistic savants, they are very good at very specific tasks, but are not very good when required to take in the larger picture. Humans, on the other hand, sometimes are autistic savants, but more generally they are more effective at large picture thinking. Thus, the best combination is to use the SOMAS framework as intelligence augmentation, focussing on how to best relay the necessary information for the operator to see the big picture and then order SOMAS to accomplish a specific task.
4. Currently, SOMAS just executes serially. However, since the bottleneck in evolutionary algorithms tends to be the fitness function, they are an embarrassingly parallel problem. Great gains in execution speed are possible by decomposing SOMAS into a grid computing solution.
5. Genetic programming can be incorporated into SOMAS by making parts of the chromosome represent an instruction set for a programming language. Since using only genetic programming can require a large amount of time to generate agent rules, a hybrid approach is possible where genetic programming is used to evolve an actuator set and the existing genetic algorithm continues to produce the weights and parameters for the sensors and rules.

6. The difference between mechanically and naturally created artifacts and intentionally created artifacts [33] based on fractal geometry can be used to develop a search heuristic that tunes search parameters based on how much human intentional development influences the search landscape.

7. The technique used in the self organization metric to extract the predictive information in a history can also be used as a form of differential evolution. Essentially, an operator uses the predictive information to predict where another good solution may lie in the solution space. The predictive information is extracted from the generative history maintained by keeping track of which chromosomes have produced each other.

8. Incorporate a lifespan into the swarm, such that very complex swarm behaviors have a short lifespan and a low likelihood of becoming chaotic, and a simple swarm behavior has a longer lifespan since it is safer.

9. Add the offline search and simulation to the containers. Thus, the containers can develop models directly from the network they reside on and can farm out chromosomes to each other for parallelized SOMAS evolution. The advantage of this approach over an offline search that is separate from the target network is a shorter response time and a smaller data transfer.

10. Allow SOMAS agents to install containers themselves. Combined with the previous item, SOMAS is self sufficient in that it does not require any kind of home-base for generation.

11. Formalize and simplify the SOMAS model. For example, the agents can be formalized to sets of rules and parameters within the container's rule and parameter sets. This helps reduce the difference between container and agent, allowing the model to be more flexible. Additionally, once the agents are represented as a set of rules, then the problem domain as a whole can be formalized very rigorously and concisely. Essentially, the domain consists of a set of containers and regions (containers within 1 hop of a container). The problem is to find the

highest ordered state transition function (as opposed to a path to goal states). A container can change its state and change the transition function in its region, though it cannot directly affect the states of other containers. Transition functions are defined by rulesets. Thus, since agents are rulesets, they define the transition functions for containers and regions. The problem becomes that of finding the necessary rulesets to create the highest ordered state transition function.

12. Behaviors can be programmed into the SOMAS agents *a priori*. For example, if the agents need to visit waypoints in the network each agent can have a list of nodes that it needs to visit. The benefit of this approach is that necessary behaviors that are already known do not have to be searched for by the swarm, and the search process can focus on the unknown aspects of the swarm behavior.

7.2.2 Testing.

1. Once ready for real world use, the self organized and entangled hierarchy metrics must be tested on very complex systems that do not have an effective means of control.
2. The entangled hierarchy metric should be used as an objective function, and compared with a priori constructed hierarchical swarms and swarm evolved without the entangled hierarchy objective function, to investigate more precisely the effect of entangled hierarchies.
3. The results can be investigated more fully by performing ANOVA tests to determine how self organization affects different objectives, and by comparing the Pareto fronts produced with and without the self organization metric. In order to do this, the metric for self organization needs to be developed so it is more accurate.
4. There are a great range of security scenarios that still need to be investigated. For instance, a network warfare competition scenario would be very interesting.

In this scenario, there are 2 or more different agent sets that all possess territory on a network. Coevolution is used to evolve the best competitive agent swarm.

Another area of exploration for network objectives is operations effictiveness. This includes capabilities such as efficient routing, information availability, providing services, etc.

Finally, continuing on the theme of human involvement discussed in the last section, it would be interesting to use human produced scenarios. One simple way of doing this is to use the Amazon© Mechanical Turk© service, where users submit a job and people all around the world complete it for a minimal fee.

5. A greater range of statistics are possible for the objective values than just the average of the simulation runs. For instance, in order to take both the number of runs and the variance into account, the confidence interval can be used. Including the number of runs in this way is useful since it allows the run count to be varied. For instance, the simulation run count can be increased until the chromosome produces a small enough confidence interval, or a run limit is passed. The statistics from this technique can also be used as an entropy metric for the general population to guage swarm stability.

7.2.3 Analysis.

1. While in multi-objective problems there is no guarantee that the solution space maps to the objective space in a regular manner, the Pareto plots suggest that this may be the case in certain instances. If so, then such a mapping can provide insight for creating specialized variation operators to exploit the mapping's pattern.
2. The Pareto plots do detail useful information about the search space. However, a sparse plot can be difficult to interpret. Using a non convex hull surfaces for PF_{Known} (i.e. not Delaunay triangulation) can make this visualization easier.

3. Self organization can be visually highlighted by making the self organized aspects of the swarm stand out more than the aspects that are not self organized. This technique is used in [117].

This appendix shows Pareto plots for each tested scenario. Since each scenario has more than 3 objectives, the Pareto plots are of a subset of the objectives in each scenario.

Numbered objectives in each scenario:

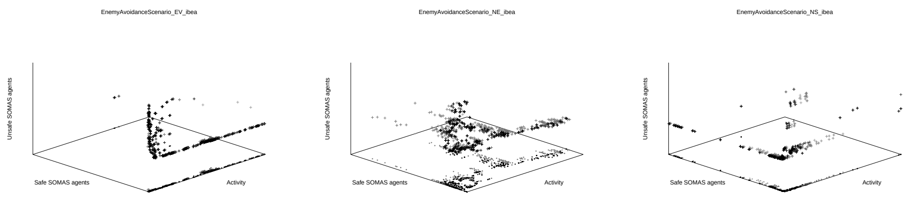
- DDoSScenario
  1. Activity
  2. Targets taken down
  3. DDos agents deleted
  4. Self organization
  5. Feasibility
- EnemyAvoidanceScenario
  1. Activity
  2. Safe somas agents
  3. Unsafe somas agents
  4. Self organization
  5. Feasibility
- IntrusionEliminationScenario
  1. Activity
  2. Locations not compromised
  3. Locations compromised
  4. Self organization
  5. Feasibility
- CompetitionScenario

    1. Activity
    2. Good agent deletion count
    3. Bad agent deletion count
    4. Self organization
    5. Feasibility

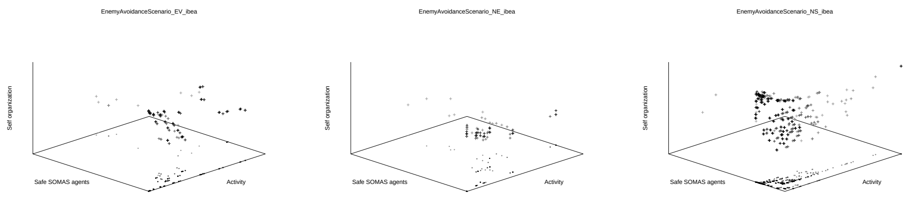
- InfoWarScenario
    1. Activity
    2. Good info
    3. Bad info
    4. Self organization
    5. Feasibility

Figure 35: EnemyAvoidanceScenario

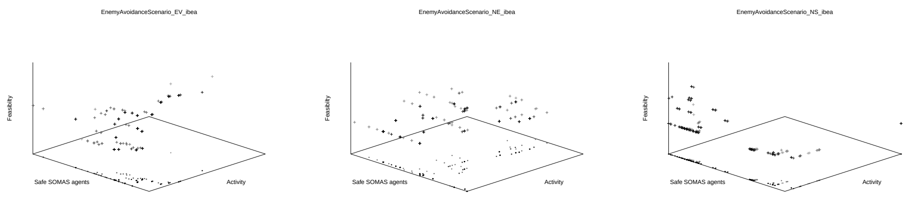
EV NE NS



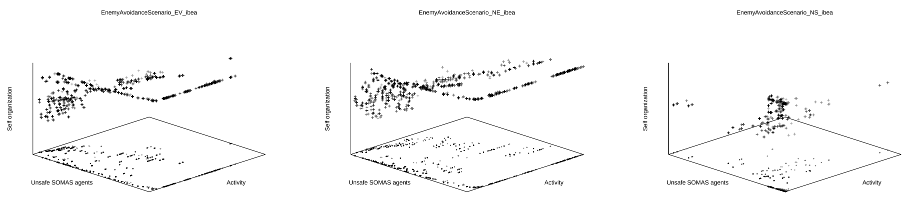
Objectives 1, 2, and 3



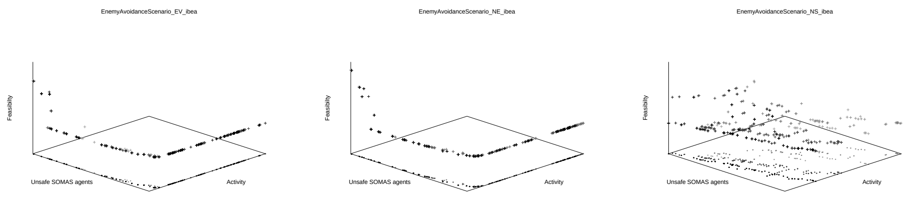
Objectives 1, 2, and 4



Objectives 1, 2, and 5



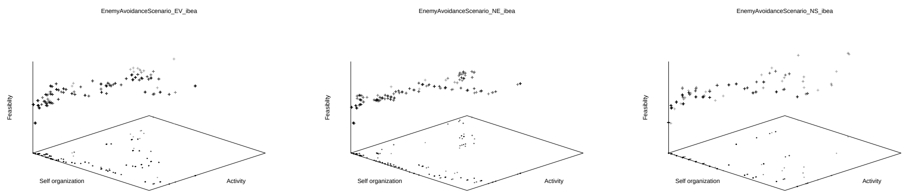
Objectives 1, 3, and 4



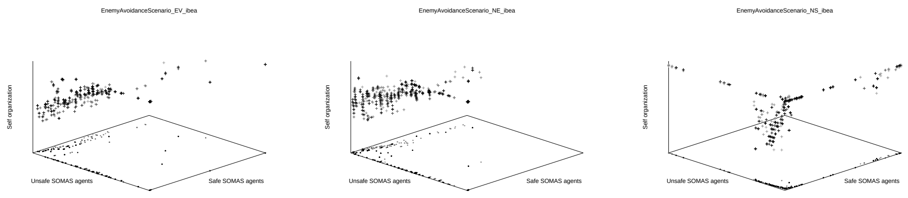
Objectives 1, 3, and 5

Figure 36: EnemyAvoidanceScenario

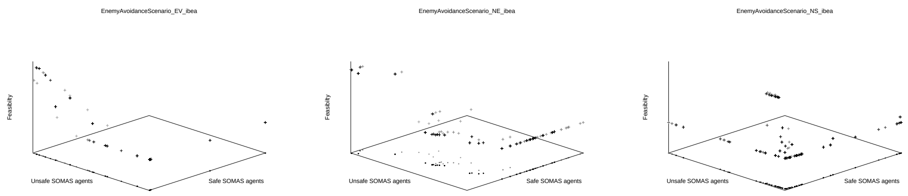
EV NE NS



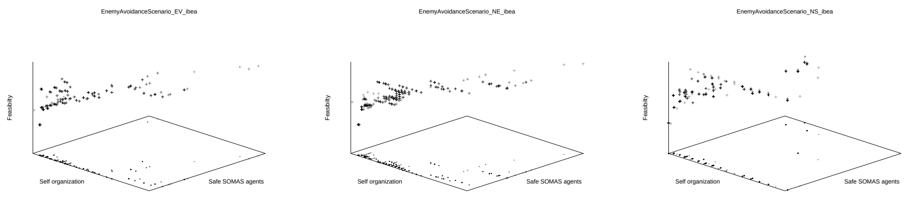
Objectives 1, 4, and 5



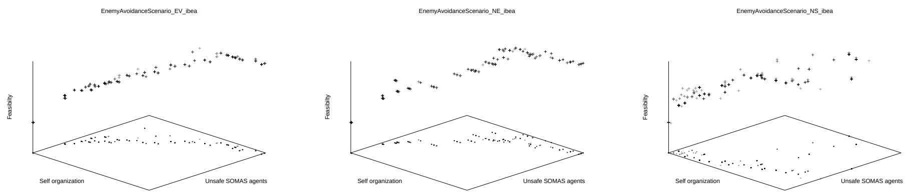
Objectives 2, 3, and 4



Objectives 2, 3, and 5



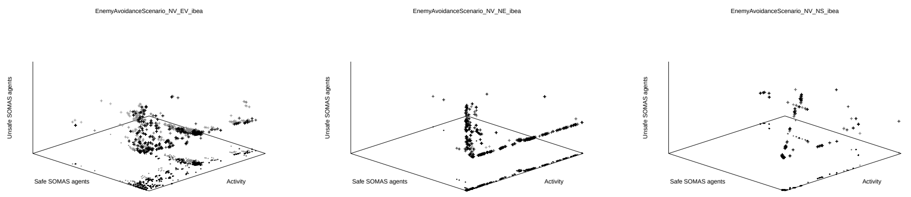
Objectives 2, 4, and 5



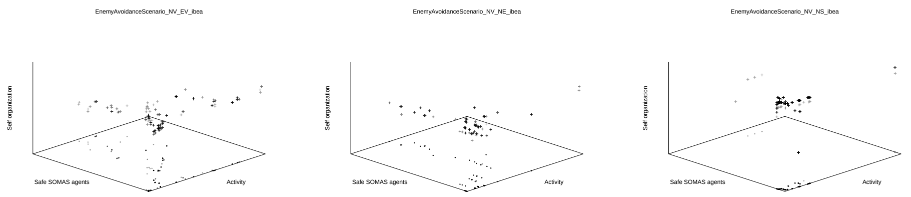
Objectives 3, 4, and 5

Figure 37: EnemyAvoidanceScenario NV

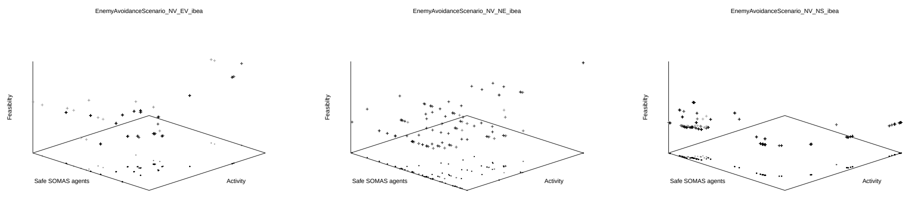
EV NE NS



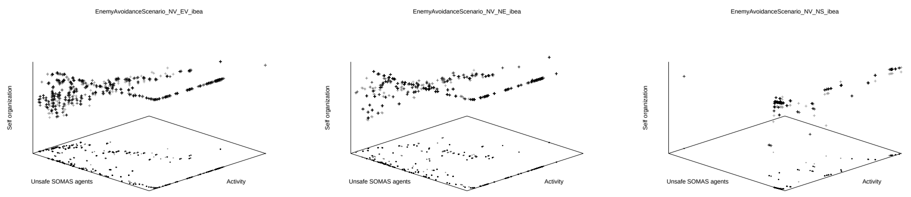
Objectives 1, 2, and 3



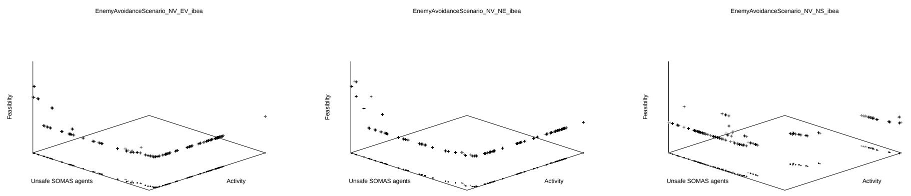
Objectives 1, 2, and 4



Objectives 1, 2, and 5



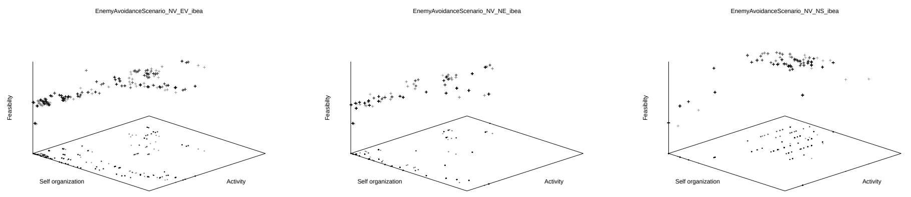
Objectives 1, 3, and 4



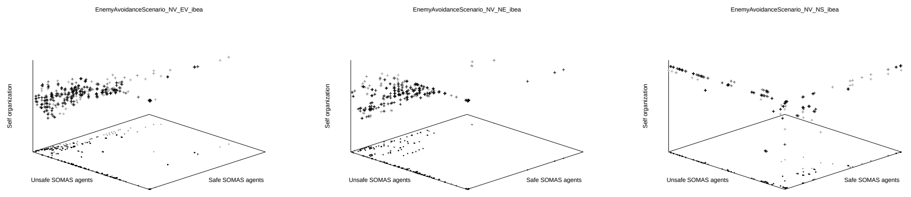
Objectives 1, 3, and 5

Figure 38: EnemyAvoidanceScenario NV

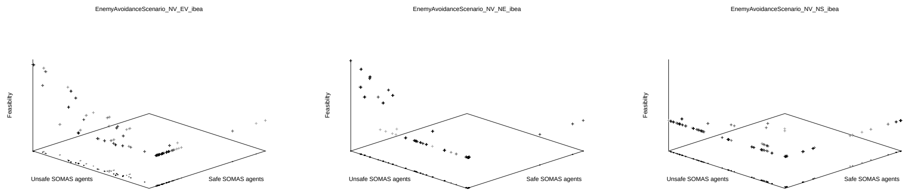
EV NE NS



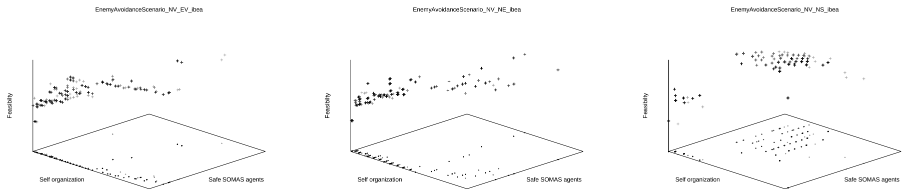
Objectives 1, 4, and 5



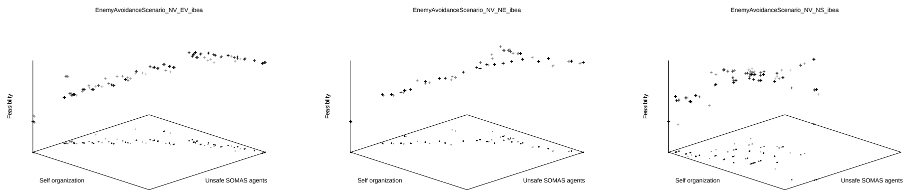
Objectives 2, 3, and 4



Objectives 2, 3, and 5



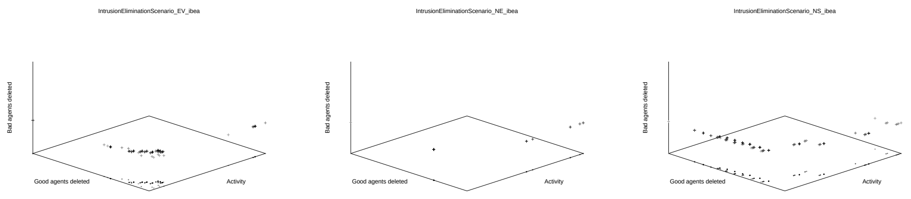
Objectives 2, 4, and 5



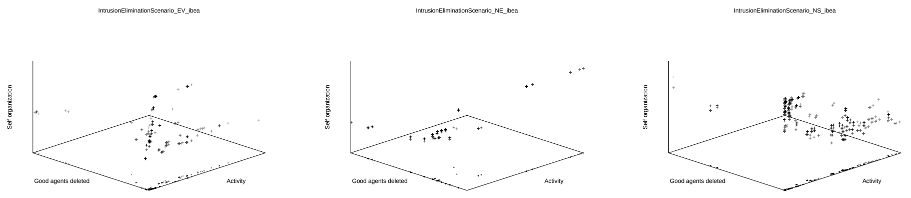
Objectives 3, 4, and 5

Figure 39: IntrusionEliminationScenario

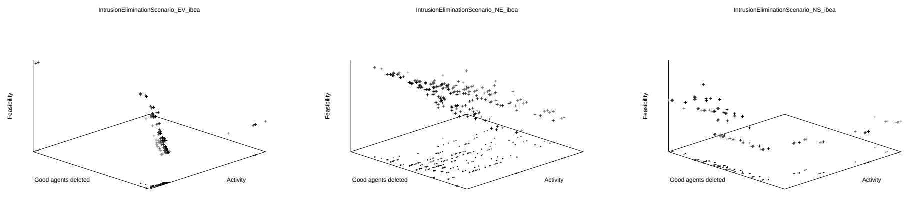
EV NE NS



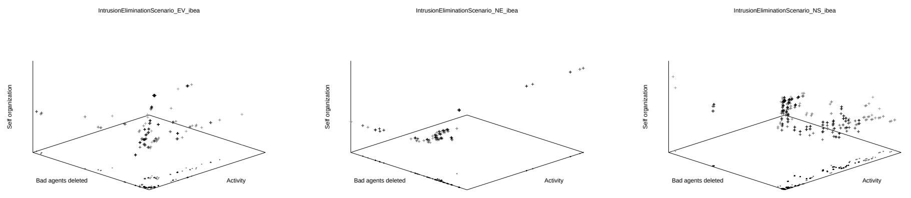
Objectives 1, 2, and 3



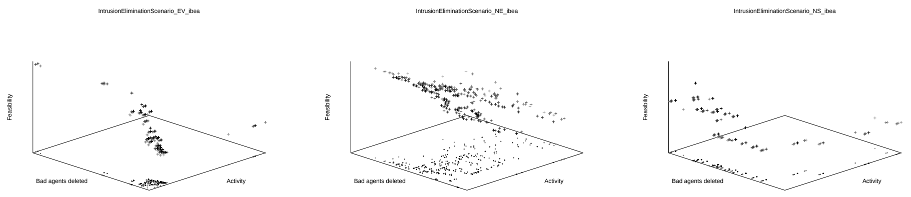
Objectives 1, 2, and 4



Objectives 1, 2, and 5



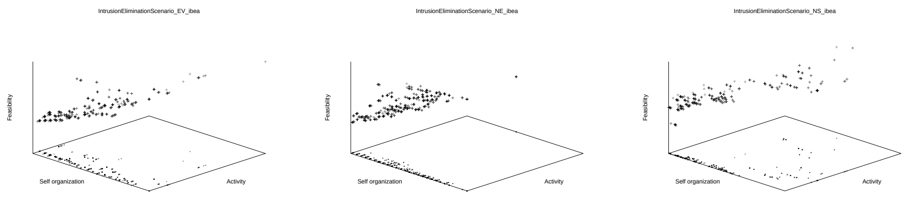
Objectives 1, 3, and 4



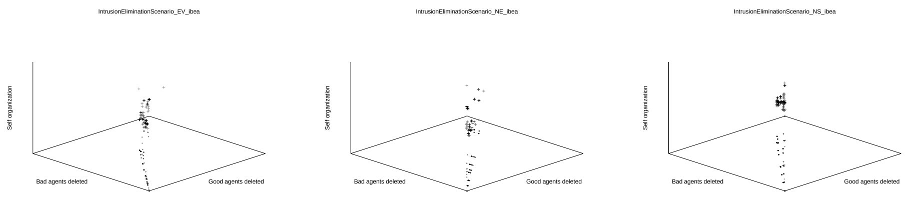
Objectives 1, 3, and 5

Figure 40: IntrusionEliminationScenario

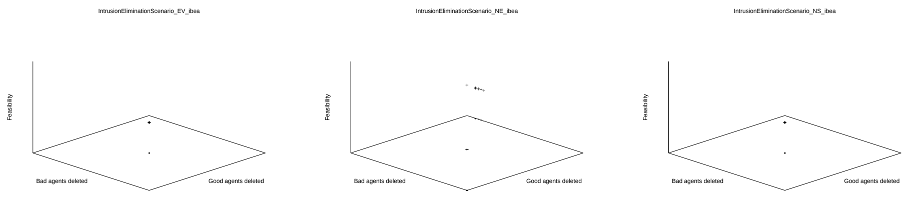
EV NE NS



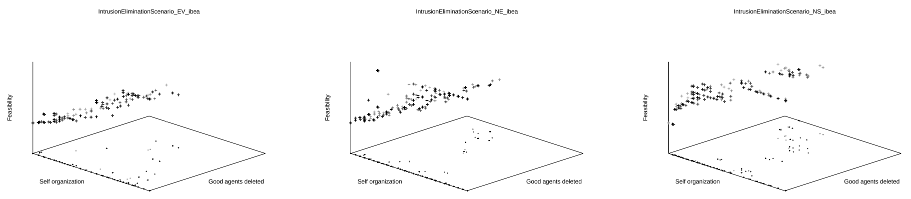
Objectives 1, 4, and 5



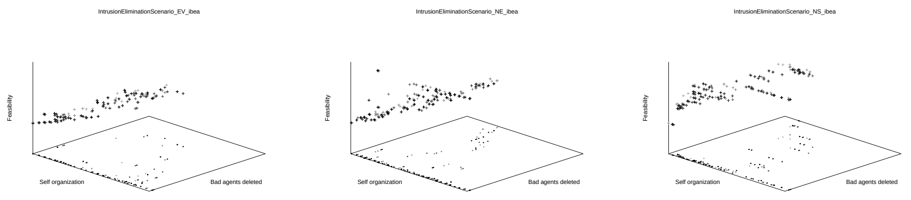
Objectives 2, 3, and 4



Objectives 2, 3, and 5



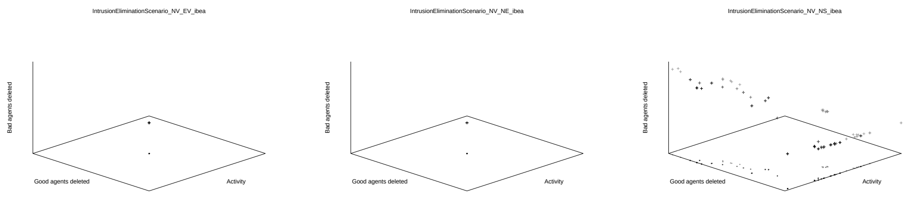
Objectives 2, 4, and 5



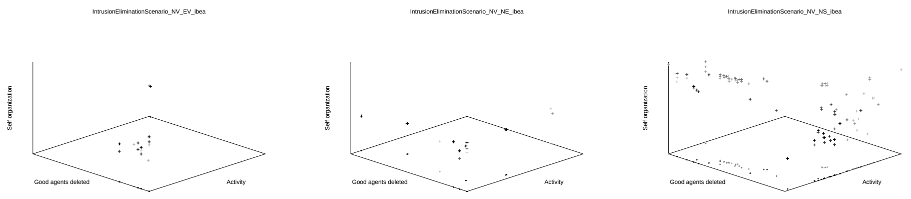
Objectives 3, 4, and 5

Figure 41: IntrusionEliminationScenario NV

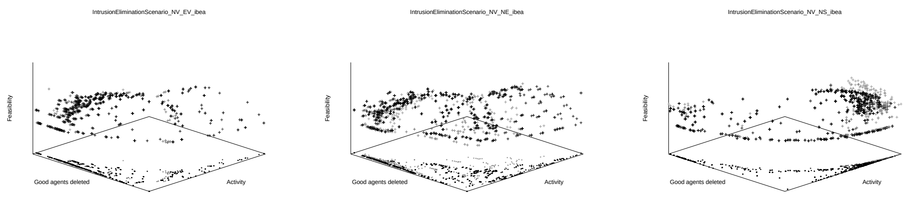
EV NE NS



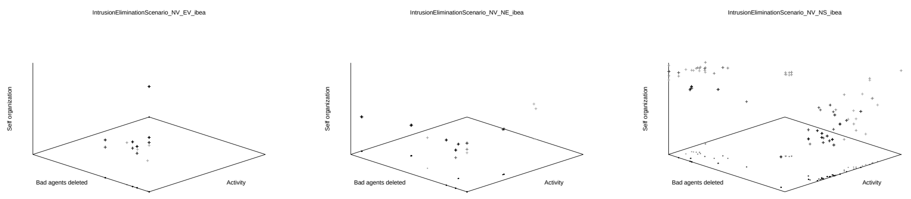
Objectives 1, 2, and 3



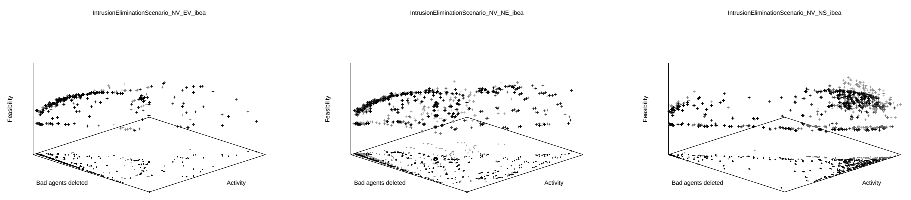
Objectives 1, 2, and 4



Objectives 1, 2, and 5



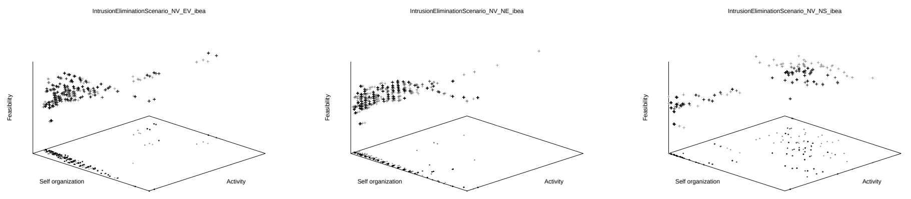
Objectives 1, 3, and 4



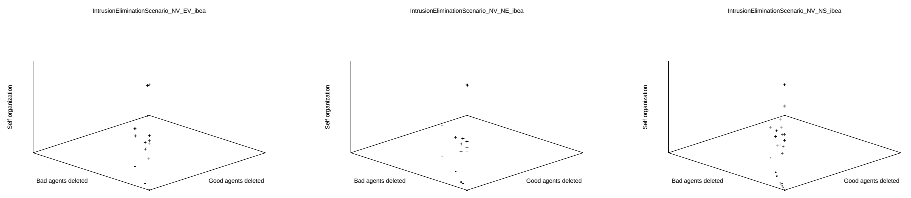
Objectives 1, 3, and 5

Figure 42: IntrusionEliminationScenario NV

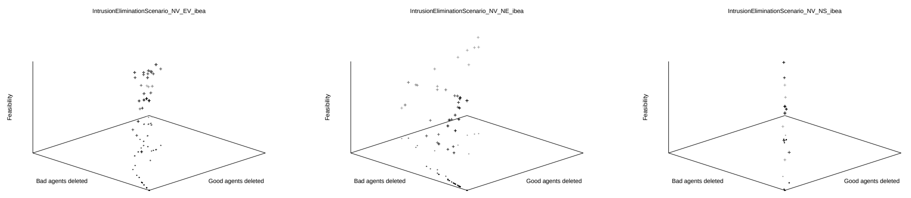
EV NE NS



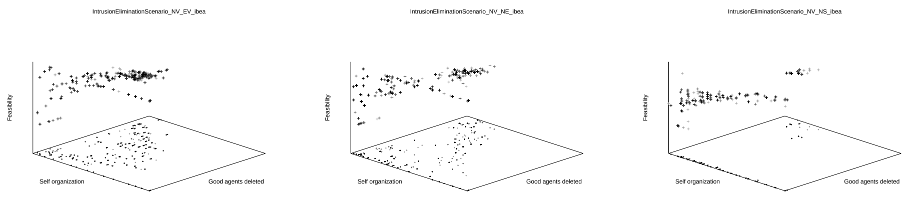
Objectives 1, 4, and 5



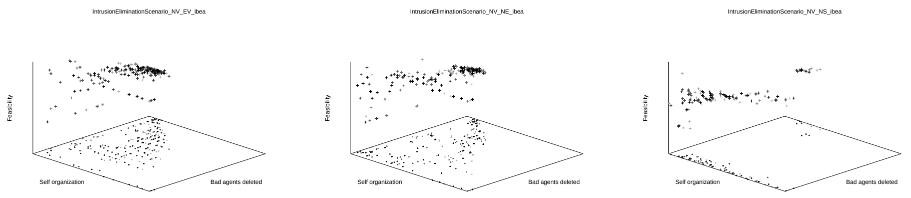
Objectives 2, 3, and 4



Objectives 2, 3, and 5



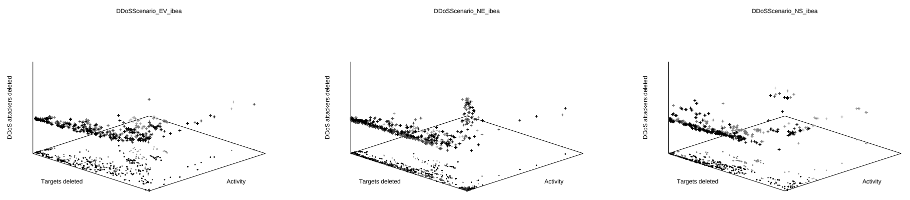
Objectives 2, 4, and 5



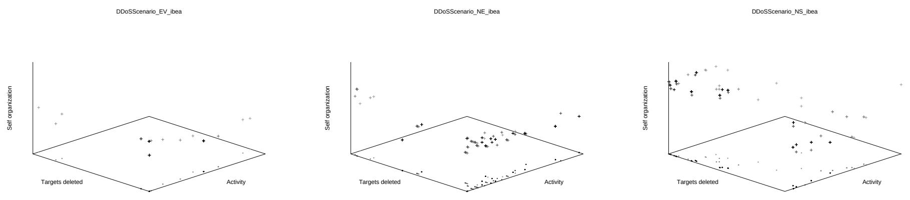
Objectives 3, 4, and 5

Figure 43: DDoSScenario

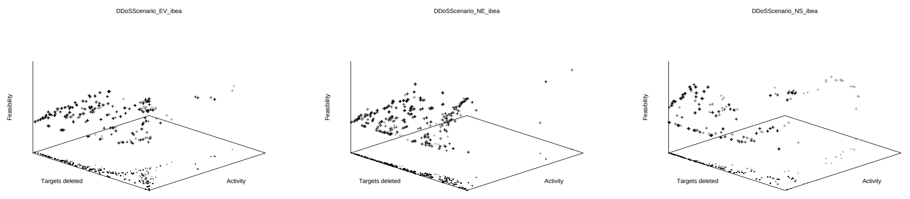
EV NE NS



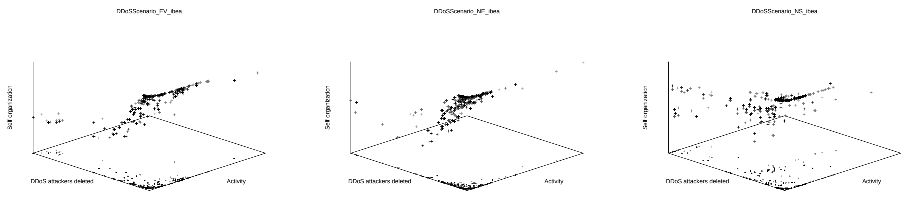
Objectives 1, 2, and 3



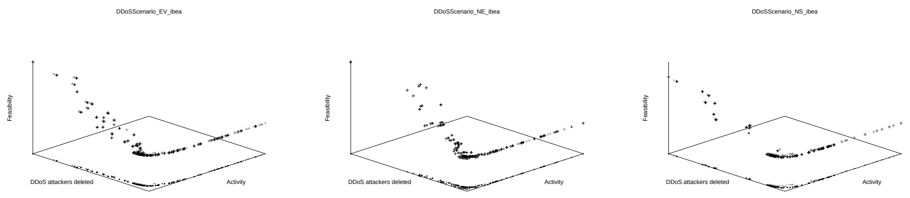
Objectives 1, 2, and 4



Objectives 1, 2, and 5



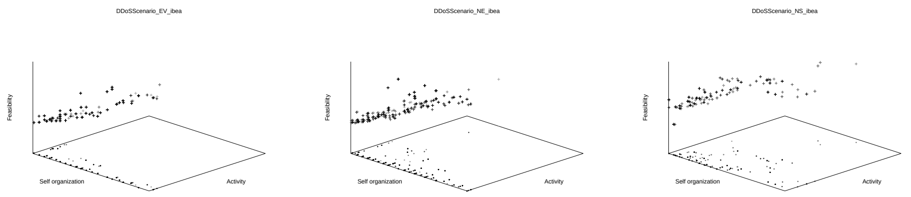
Objectives 1, 3, and 4



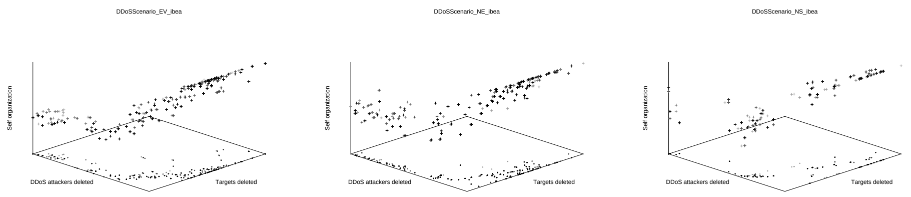
Objectives 1, 3, and 5

Figure 44: DDoSScenario

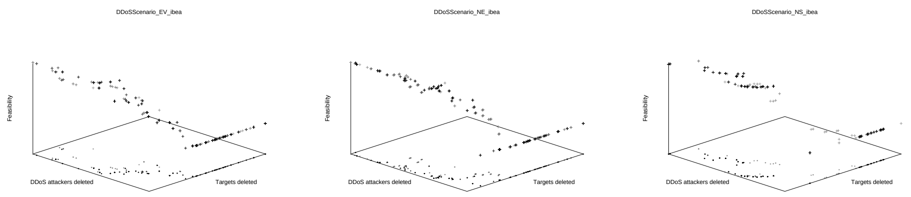
EV NE NS



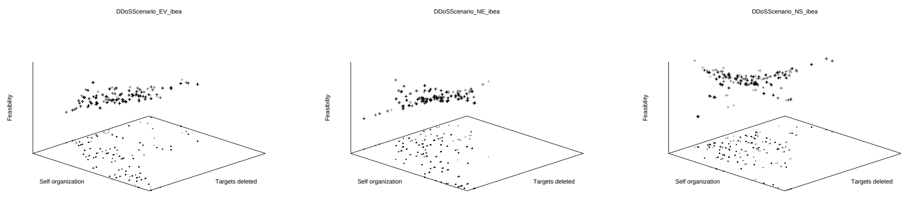
Objectives 1, 4, and 5



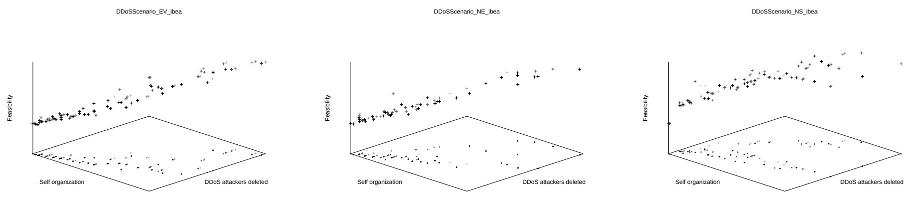
Objectives 2, 3, and 4



Objectives 2, 3, and 5



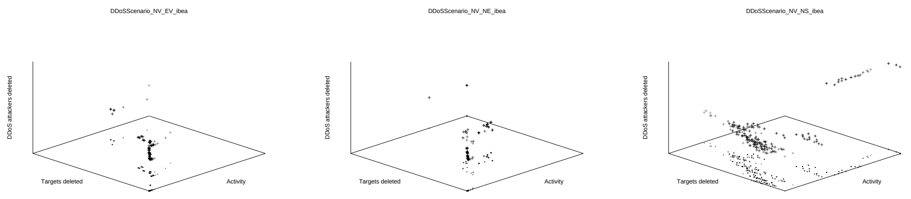
Objectives 2, 4, and 5



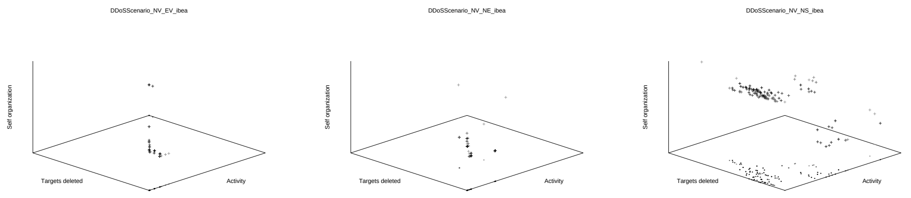
Objectives 3, 4, and 5

Figure 45: DDoSScenario NV

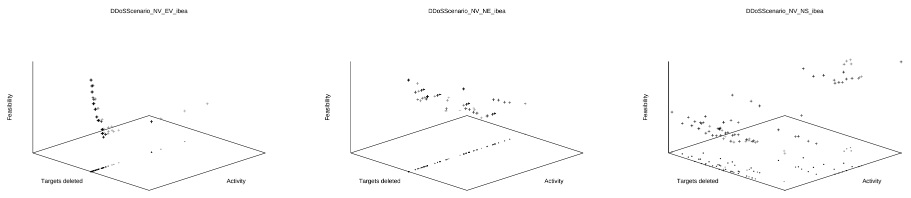
EV NE NS



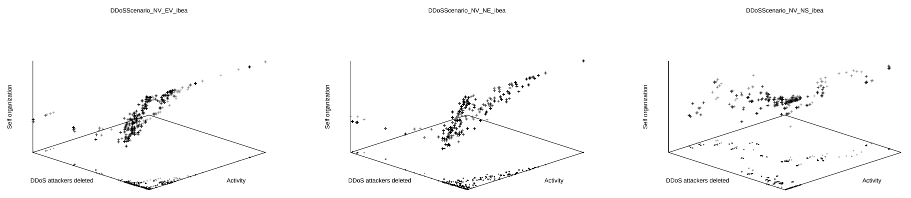
Objectives 1, 2, and 3



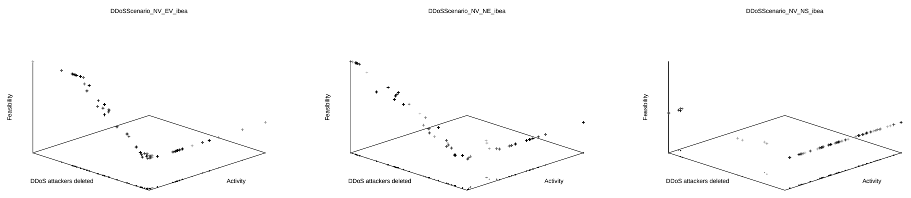
Objectives 1, 2, and 4



Objectives 1, 2, and 5



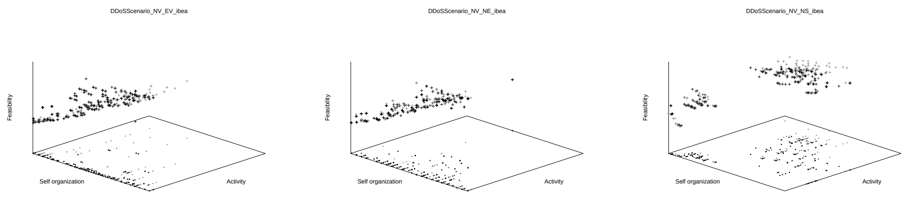
Objectives 1, 3, and 4



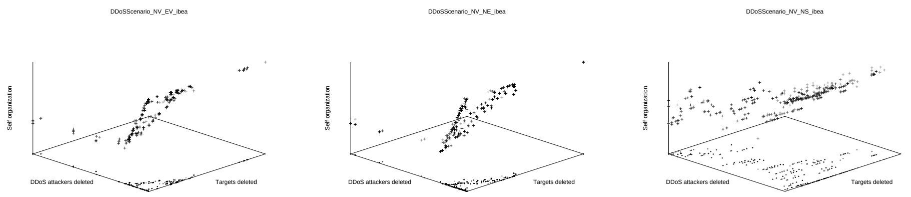
Objectives 1, 3, and 5

Figure 46: DDoSScenario NV

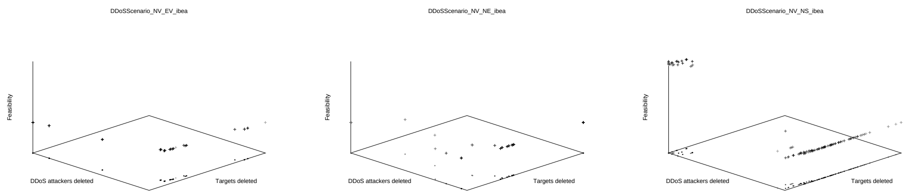
EV NE NS



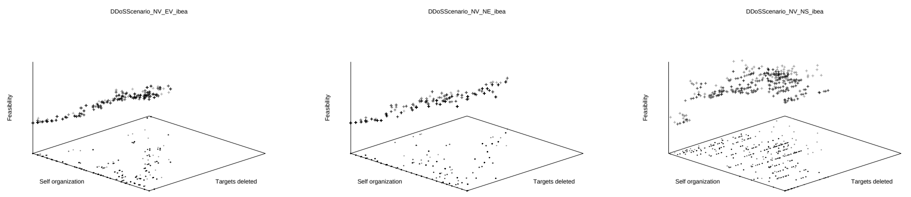
Objectives 1, 4, and 5



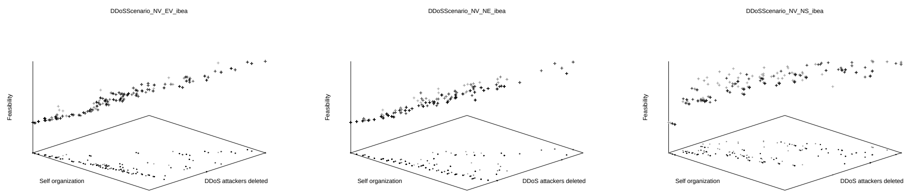
Objectives 2, 3, and 4



Objectives 2, 3, and 5



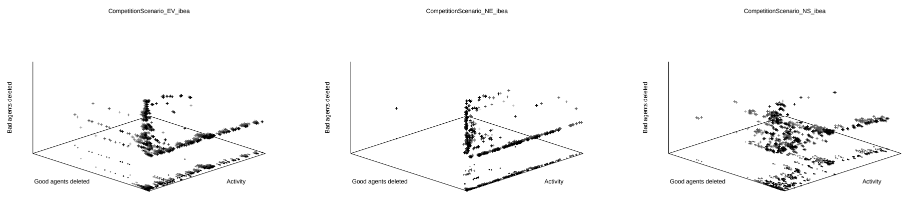
Objectives 2, 4, and 5



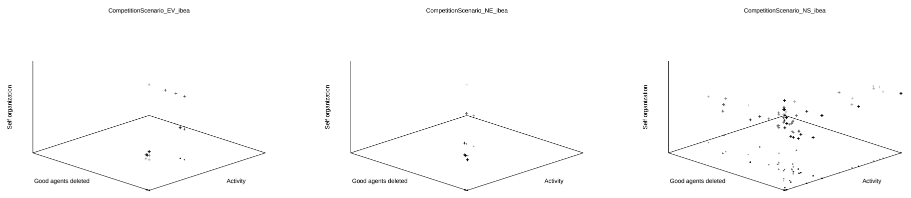
Objectives 3, 4, and 5

Figure 47: CompetitionScenario

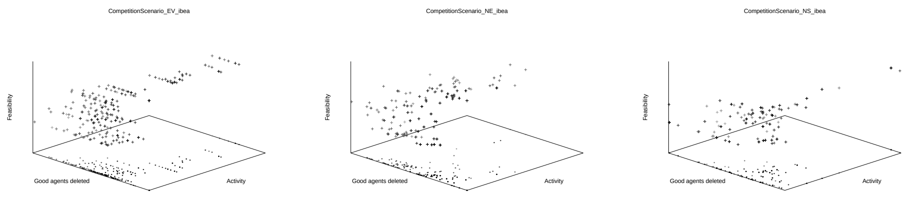
EV NE NS



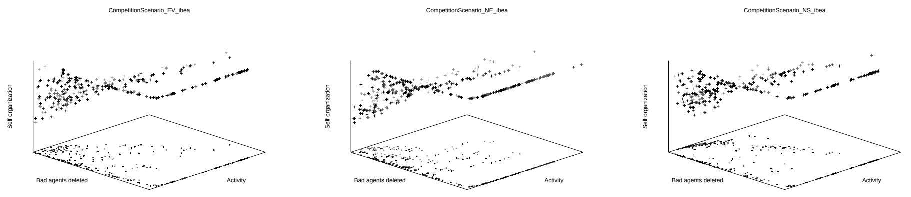
Objectives 1, 2, and 3



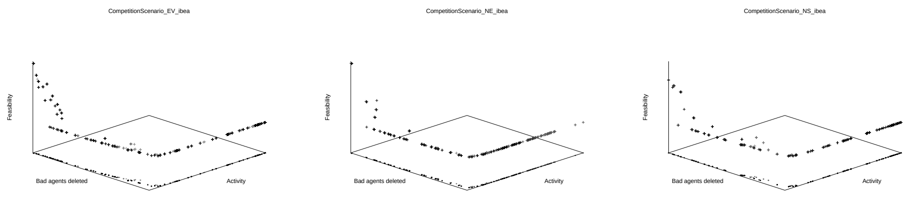
Objectives 1, 2, and 4



Objectives 1, 2, and 5



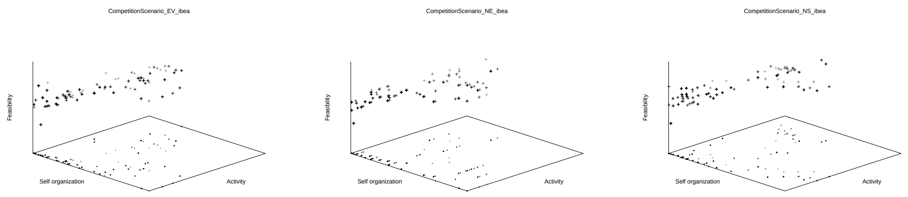
Objectives 1, 3, and 4



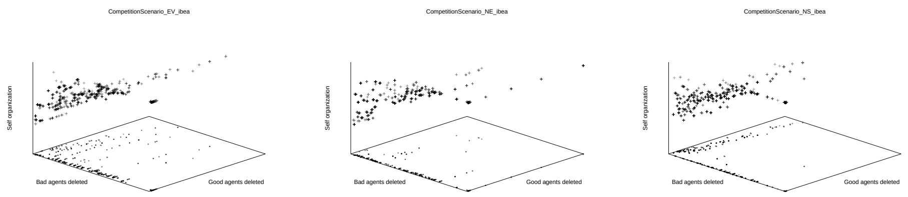
Objectives 1, 3, and 5

Figure 48: CompetitionScenario

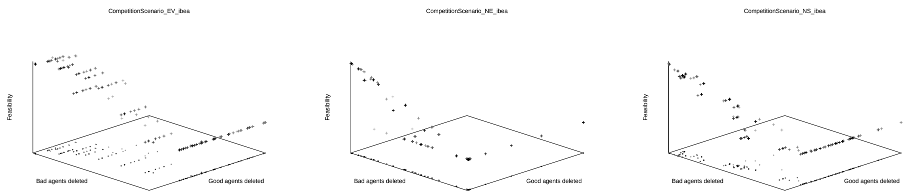
EV NE NS



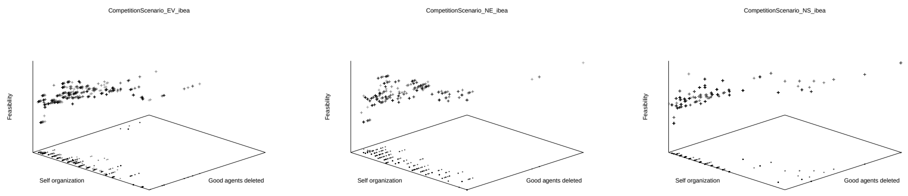
Objectives 1, 4, and 5



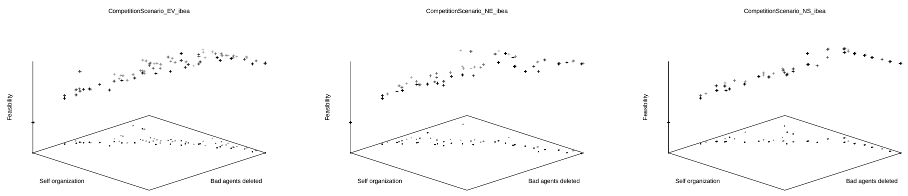
Objectives 2, 3, and 4



Objectives 2, 3, and 5



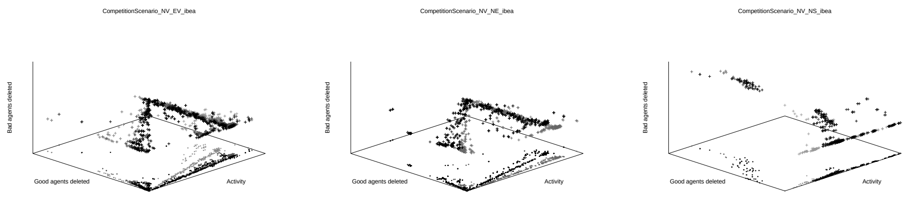
Objectives 2, 4, and 5



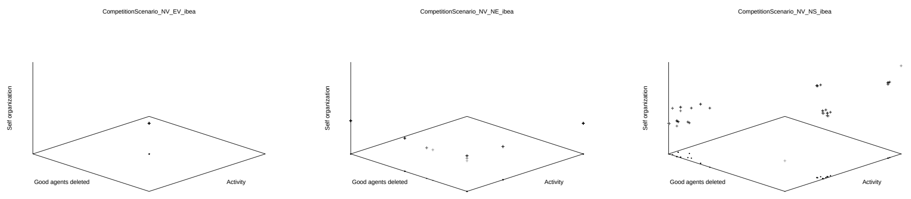
Objectives 3, 4, and 5

Figure 49: CompetitionScenario NV

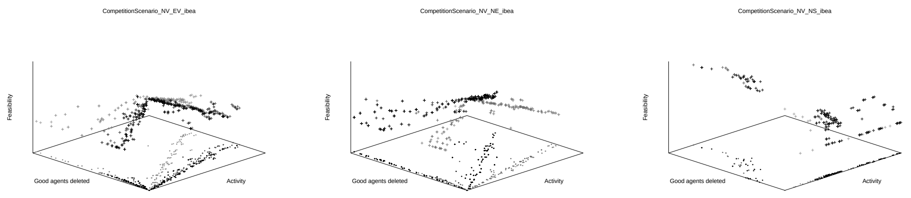
EV NE NS



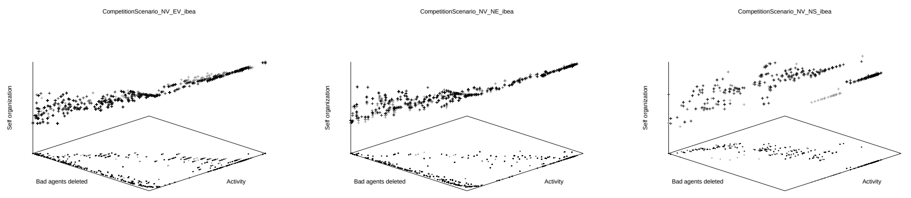
Objectives 1, 2, and 3



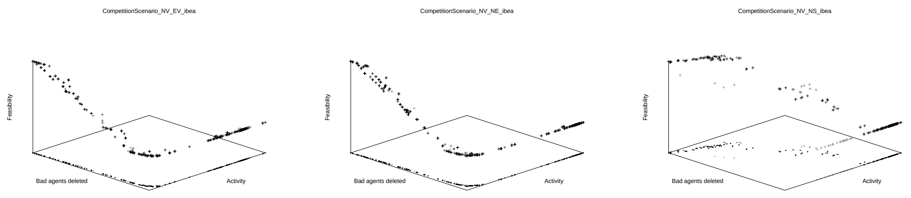
Objectives 1, 2, and 4



Objectives 1, 2, and 5



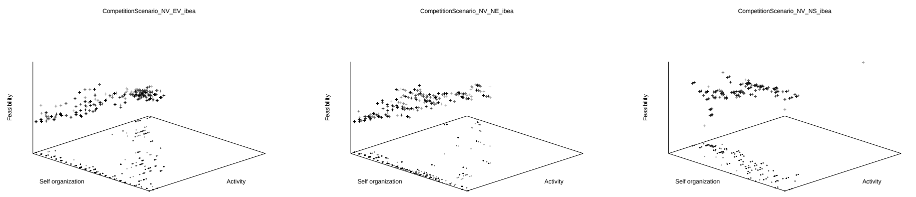
Objectives 1, 3, and 4



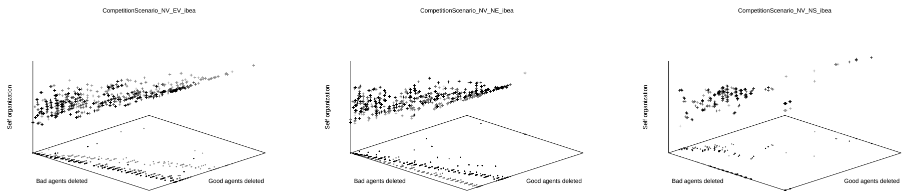
Objectives 1, 3, and 5

Figure 50: CompetitionScenario NV

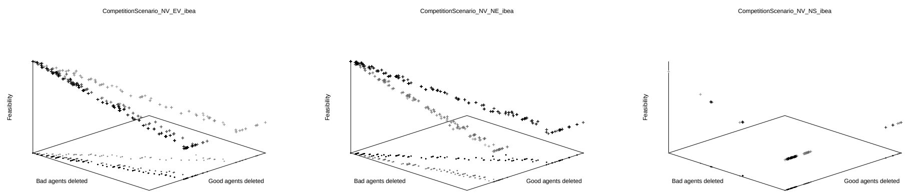
EV NE NS



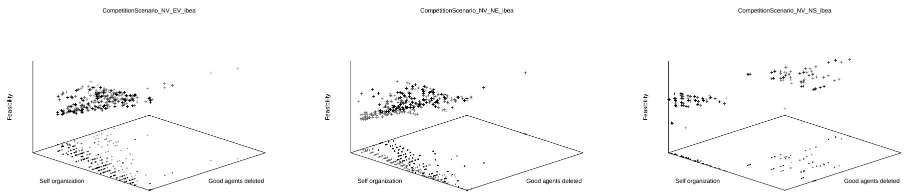
Objectives 1, 4, and 5



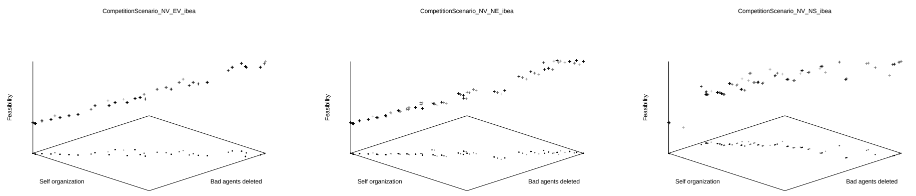
Objectives 2, 3, and 4



Objectives 2, 3, and 5



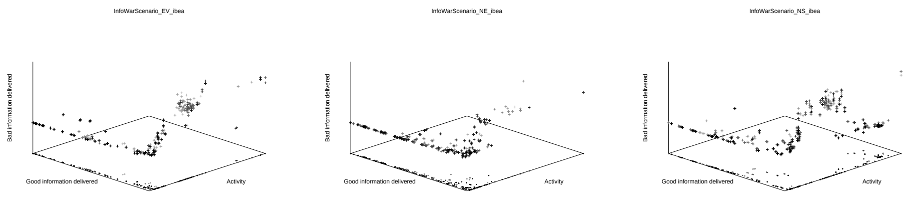
Objectives 2, 4, and 5



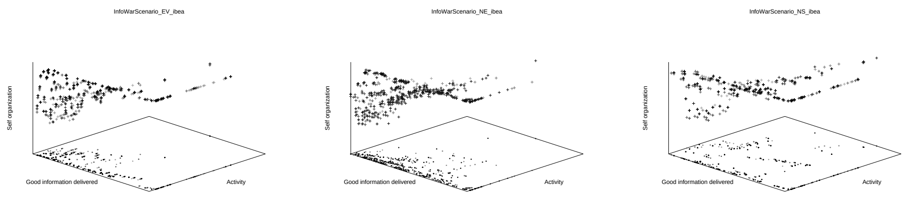
Objectives 3, 4, and 5

Figure 51: InfoWarScenario

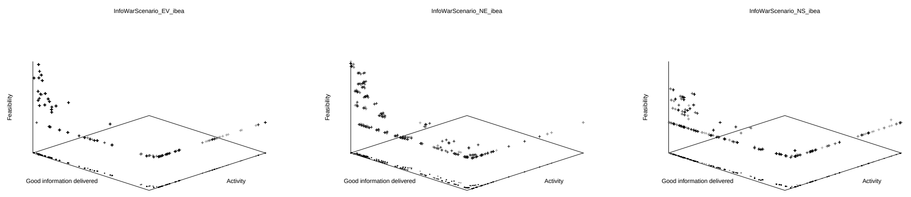
EV NE NS



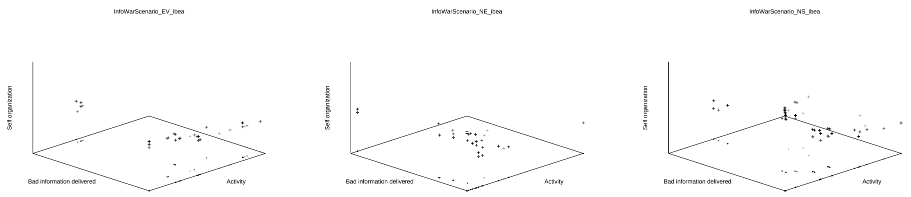
Objectives 1, 2, and 3



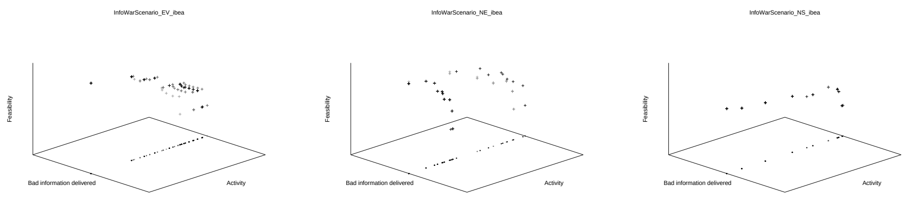
Objectives 1, 2, and 4



Objectives 1, 2, and 5



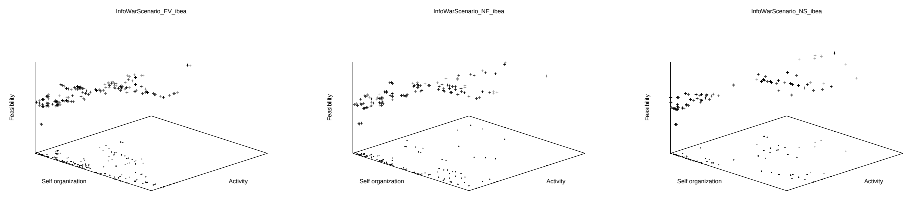
Objectives 1, 3, and 4



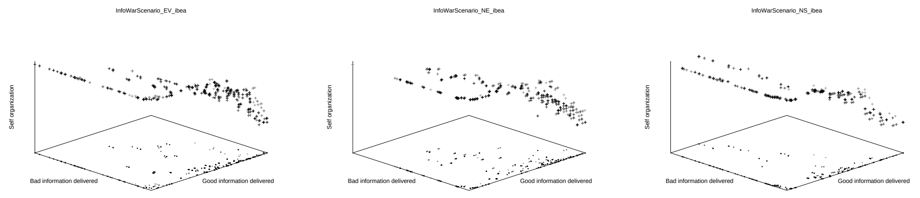
Objectives 1, 3, and 5

Figure 52: InfoWarScenario

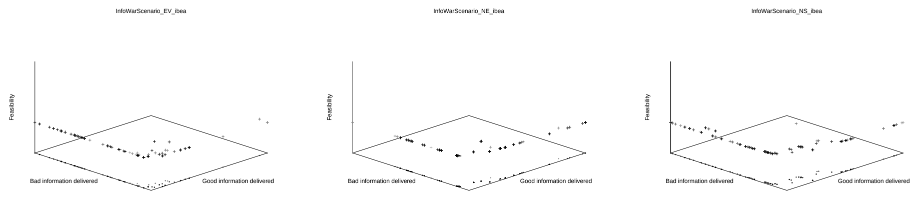
EV NE NS



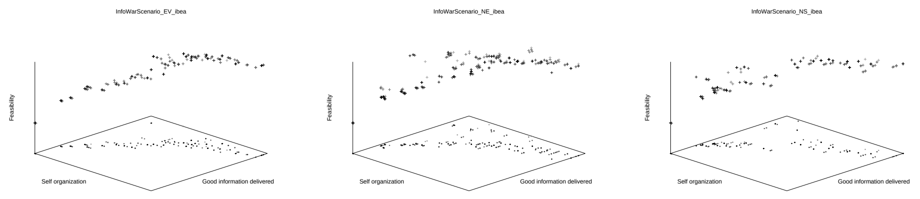
Objectives 1, 4, and 5



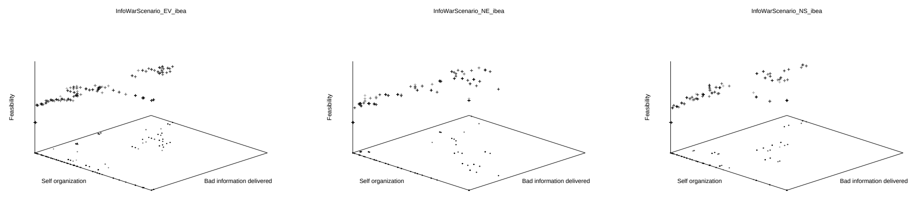
Objectives 2, 3, and 4



Objectives 2, 3, and 5



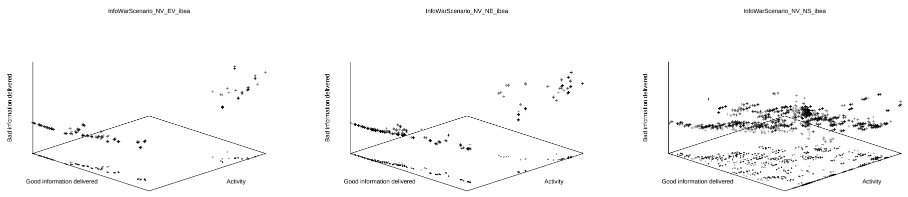
Objectives 2, 4, and 5



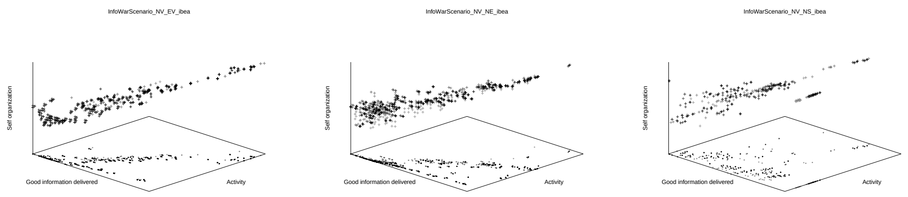
Objectives 3, 4, and 5

Figure 53: InfoWarScenario NV

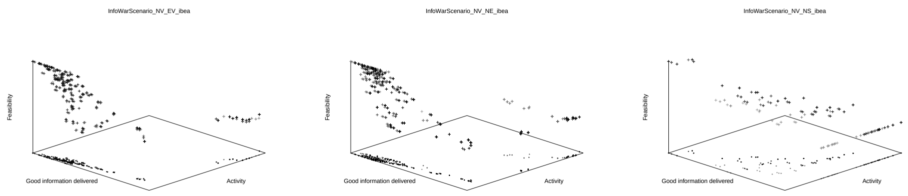
EV NE NS



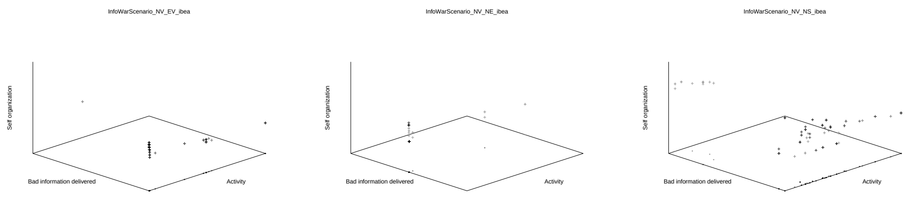
Objectives 1, 2, and 3



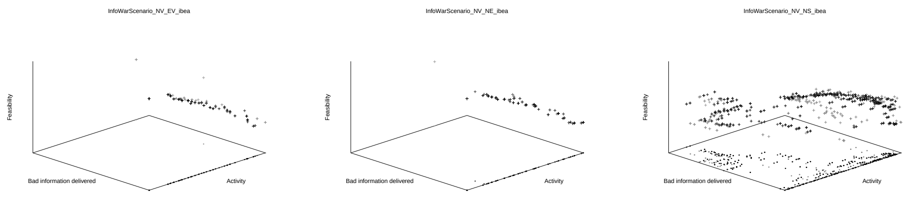
Objectives 1, 2, and 4



Objectives 1, 2, and 5



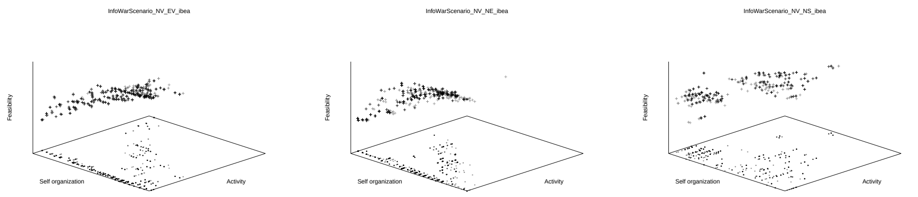
Objectives 1, 3, and 4



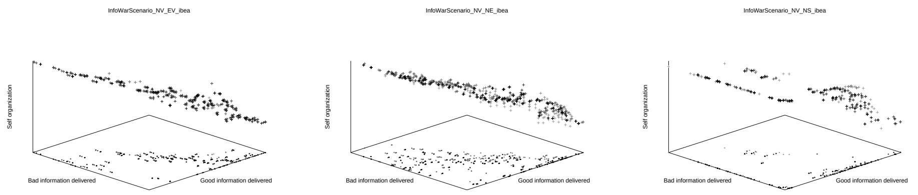
Objectives 1, 3, and 5

Figure 54: InfoWarScenario NV

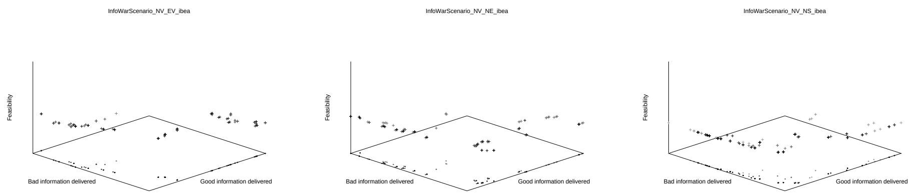
EV NE NS



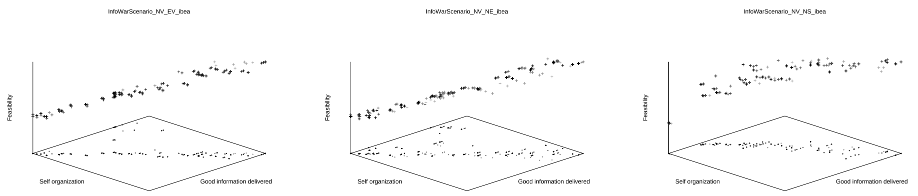
Objectives 1, 4, and 5



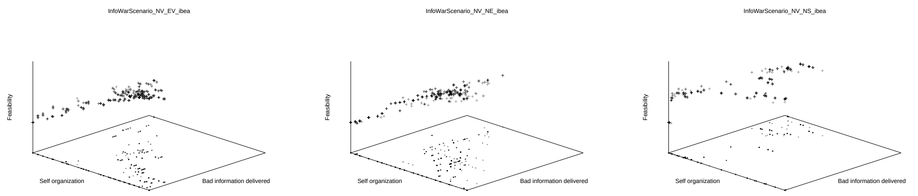
Objectives 2, 3, and 4



Objectives 2, 3, and 5

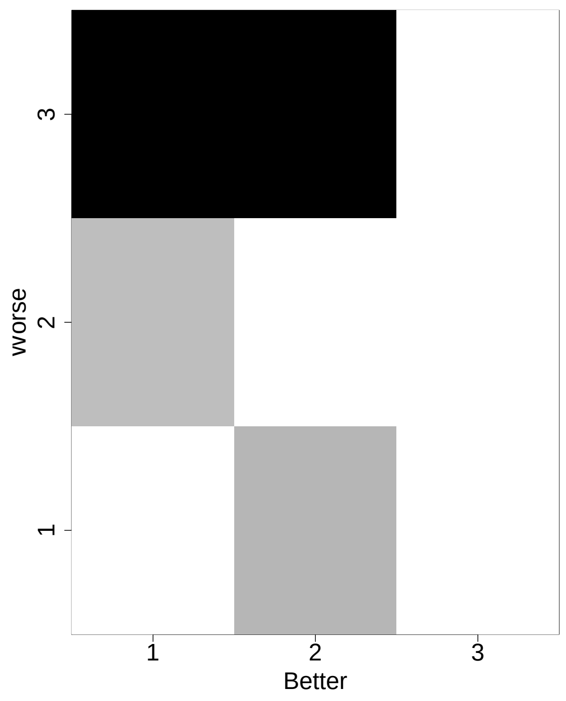


Objectives 2, 4, and 5



Objectives 3, 4, and 5

Figure 55: Interpreting results diagrams



Each grid cell is the p-value of a significance test. If the cell is completely black, then the a value will have to have been set < 0.05 in order to reject H0.
For the Fisher sign and Mann-Whitney tests the hypotheses are:

- H0: $M_{\text{better}} = M_{\text{worse}}$
- H1: $M_{\text{better}} < M_{\text{worse}}$

Where M is the population median of the Pareto metric calculated on 5 samples. A lower metric value is better than a higher metric value.
The correspondence between axis numbers and swarm types is:

1. EV: SOMAS swarm uses online evolution
2. NE: SOMAS swarm uses online search without evolution
3. NS: SOMAS swarm does not use online search

Example: the above image shows that both EV and NE outperform NS with statistical significance, whereas they are indifferent when compared to each other.

Table 20: Effectiveness of online search to accomplish swarm behaviors with network deactivation and activation actuators

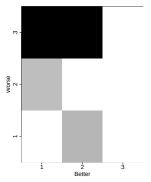
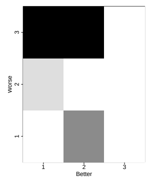
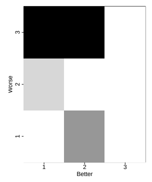
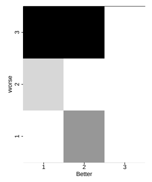
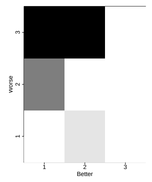
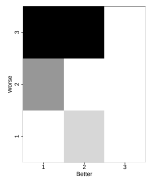
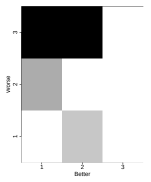
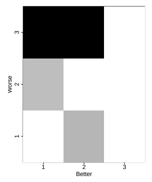
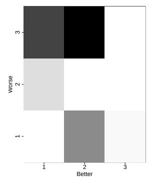
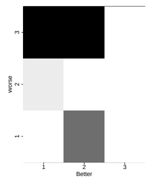
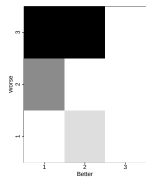
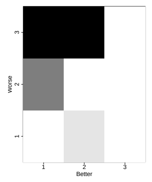
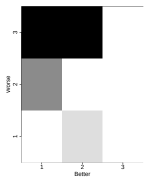
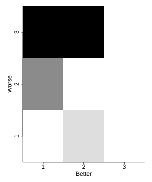
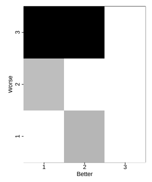
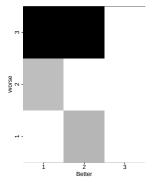
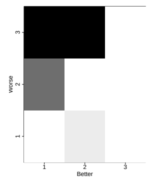
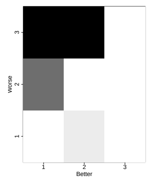
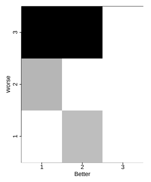
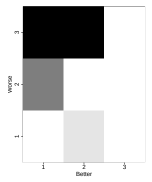
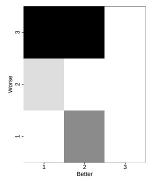
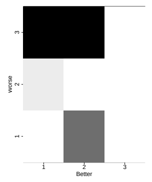
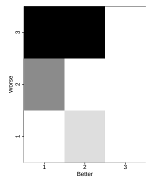
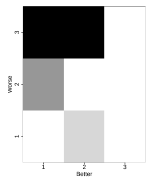
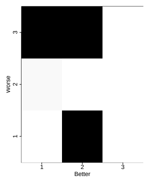
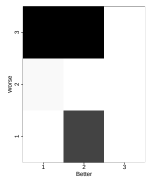
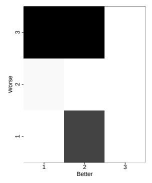
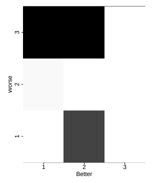
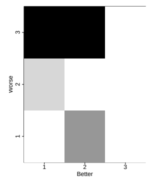
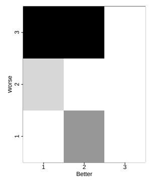
Behavior	Epsilon Metric		Hypervolume Metric		R_2 Metric	
	Fisher sign test	Mann-Whitney test	Fisher sign test	Mann-Whitney test	Fisher sign test	Mann-Whitney test
Enemy Avoidance						
Intrusion Elimination						
DDoS Protection						
Competition						
Information Protection						

Table 21: Effectiveness of online search to accomplish swarm behaviors without network deactivation and reactivation actuators

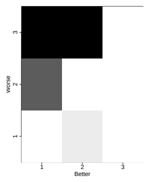
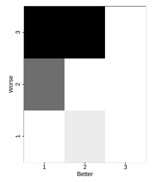
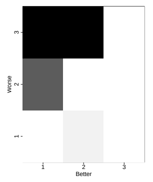
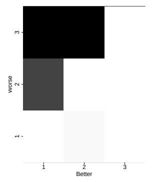
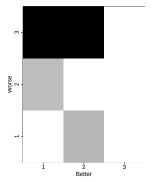
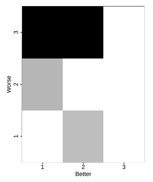
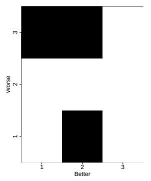
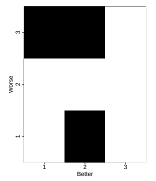
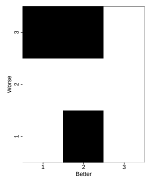
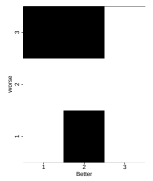
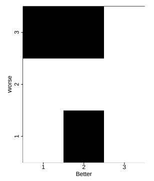
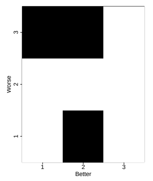
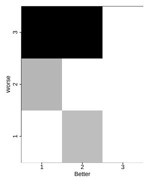
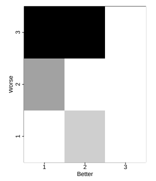
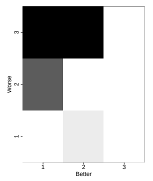
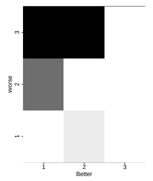
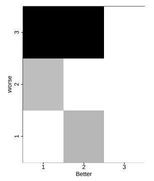
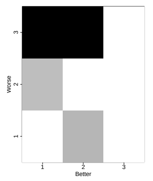
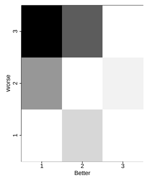
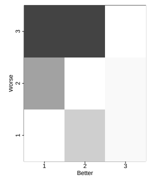
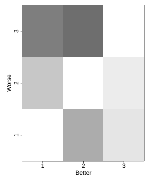
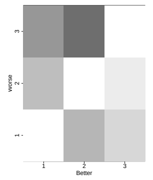
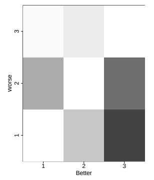
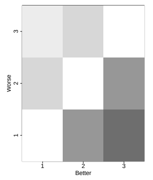
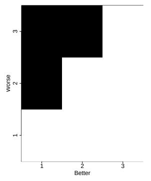
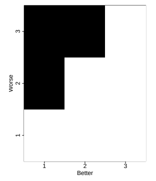
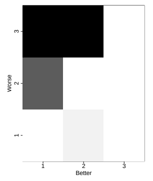
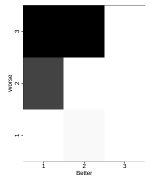
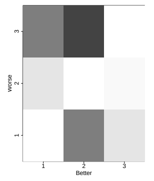
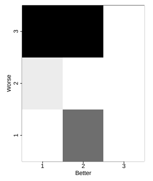
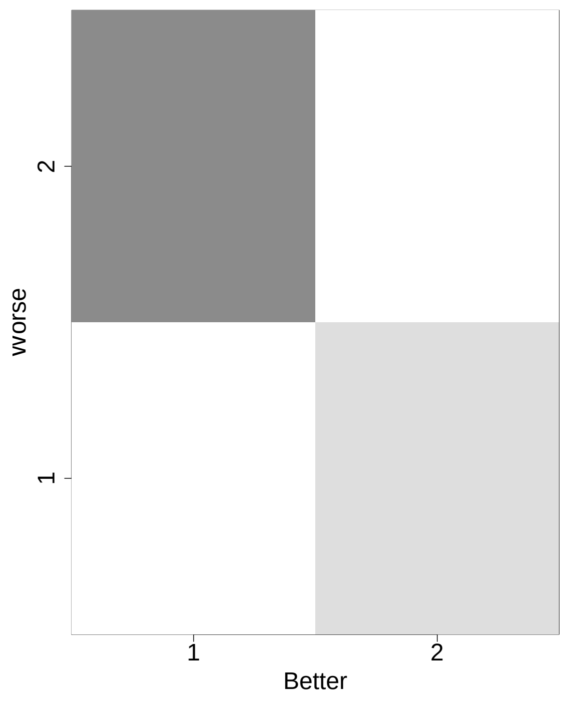
Behavior	Epsilon Metric		Hypervolume Metric		R_2 Metric	
	Fisher sign test	Mann-Whitney test	Fisher sign test	Mann-Whitney test	Fisher sign test	Mann-Whitney test
Enemy Avoidance						
Intrusion Elimination						
DDoS Protection						
Competition						
Information Protection						

Figure 56: Interpreting results diagrams



Each grid cell is the p-value of a significance test. If the cell is completely black, then the a value will have to have been set < 0.05 in order to reject H0.
For the Fisher sign and Mann-Whitney tests the hypotheses are:

- H0: $M_{\text{better}} = M_{\text{worse}}$
- H1: $M_{\text{better}} < M_{\text{worse}}$

Where M is the population median of the Pareto metric calculated on 5 samples. A lower metric value is better than a higher metric value.
The correspondence between axis numbers and swarm types is:

1. V: SOMAS swarm has node deactivation and reactivation actuators
2. NV: SOMAS swarm does not have node deactivation and reactivation actuators

Example: the above image shows that V statistically outperforms NV, but not with statistical significance.

Table 22: Enemy Avoidance scenario network destruction experiments

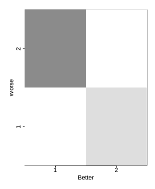
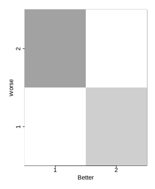
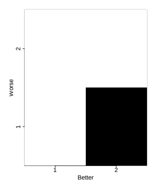
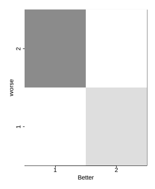
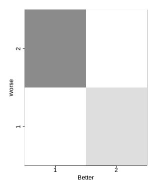
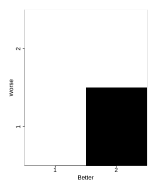
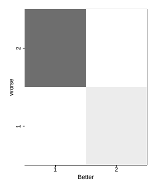
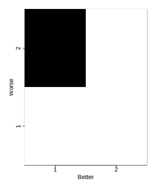
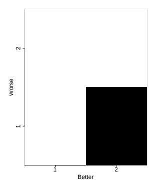
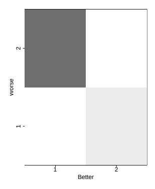
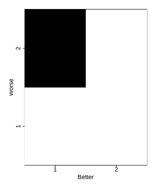
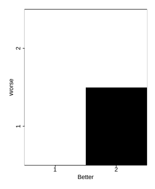
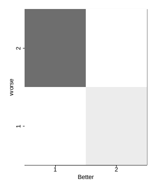
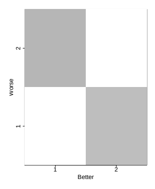
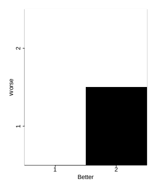
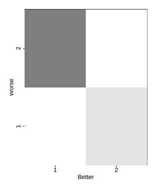
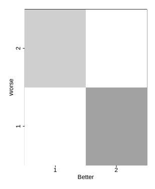
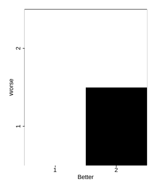
Metric	Statistical Test	EV	NE	NS
Epsilon	Fisher sign			
	Mann-Whitney			
Hypervolume	Fisher sign			
	Mann-Whitney			
R_2	Fisher sign			
	Mann-Whitney			

Table 23: Intrusion Elimination scenario network destruction experiments

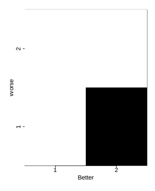
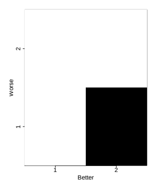
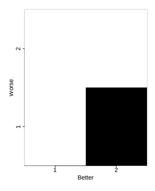
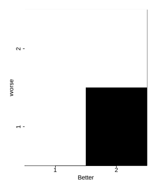
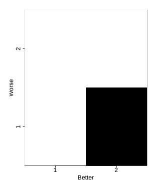
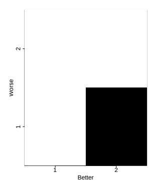
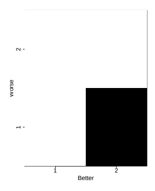
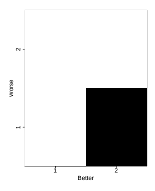
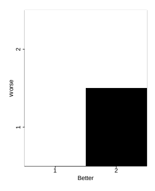
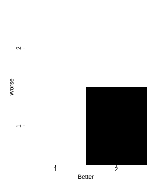
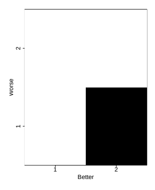
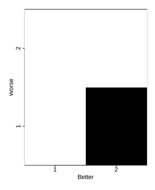
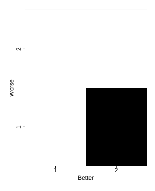
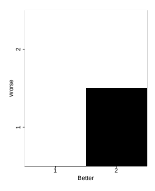
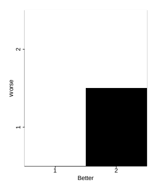
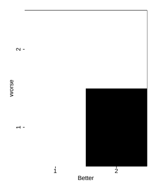
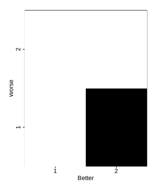
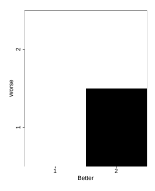
Metric	Statistical Test	EV	NE	NS
Epsilon	Fisher sign			
	Mann-Whitney			
Hypervolume	Fisher sign			
	Mann-Whitney			
R_2	Fisher sign			
	Mann-Whitney			

Table 24: DDoS scenario network destruction experiments

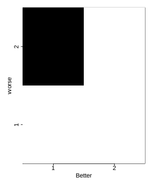
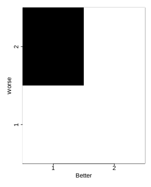
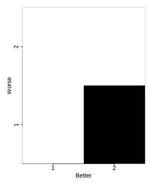
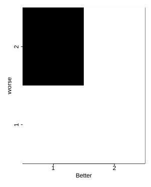
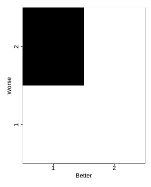
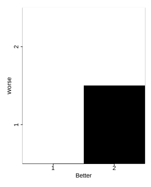
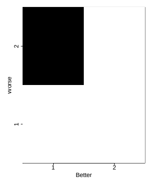
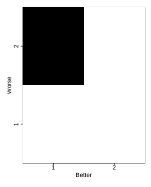
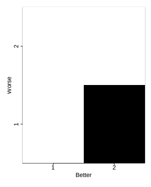
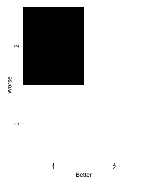
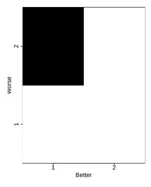
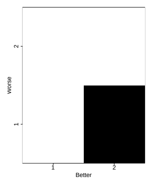
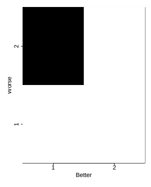
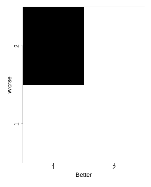
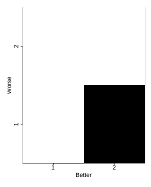
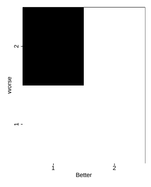
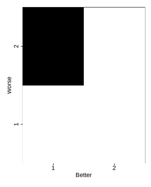
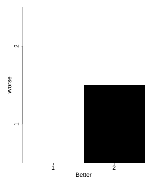
Metric	Statistical Test	EV	NE	NS
Epsilon	Fisher sign			
	Mann-Whitney			
Hypervolume	Fisher sign			
	Mann-Whitney			
R_2	Fisher sign			
	Mann-Whitney			

Table 25: Competition scenario network destruction experiments

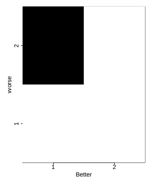
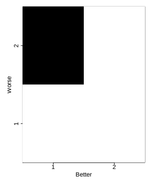
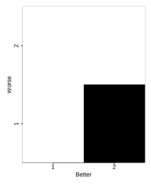
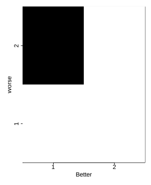
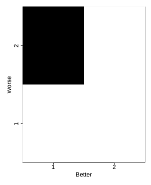
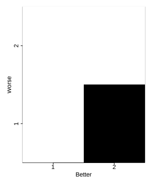
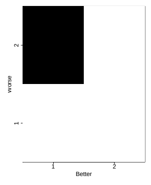
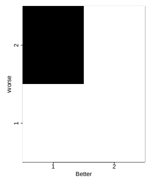
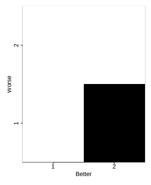
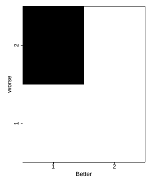
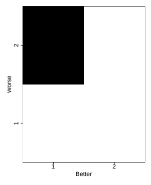
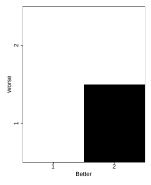
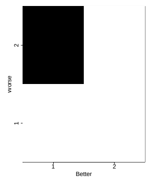
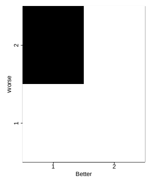
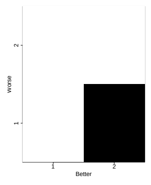
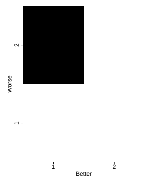
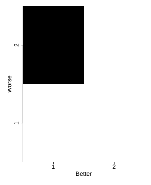
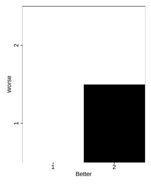
Metric	Statistical Test	EV	NE	NS
Epsilon	Fisher sign			
	Mann-Whitney			
Hypervolume	Fisher sign			
	Mann-Whitney			
R_2	Fisher sign			
	Mann-Whitney			

Table 26: Info War scenario network destruction experiments

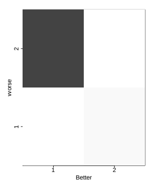
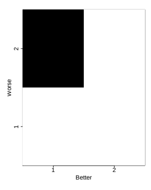
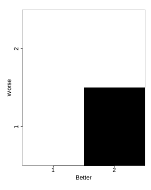
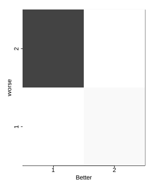
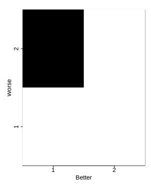
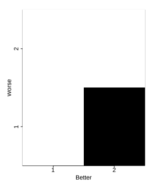
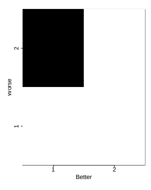
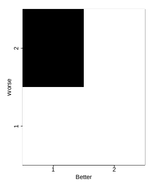
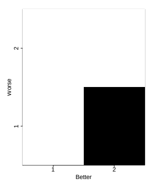
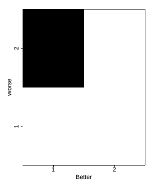
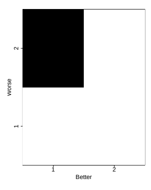
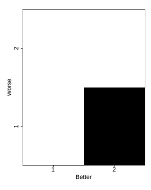
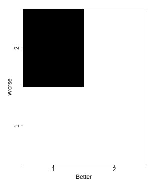
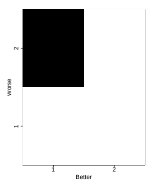
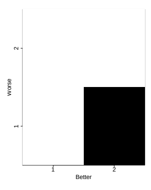
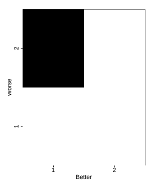
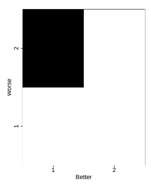
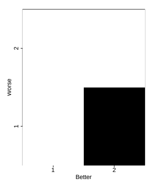
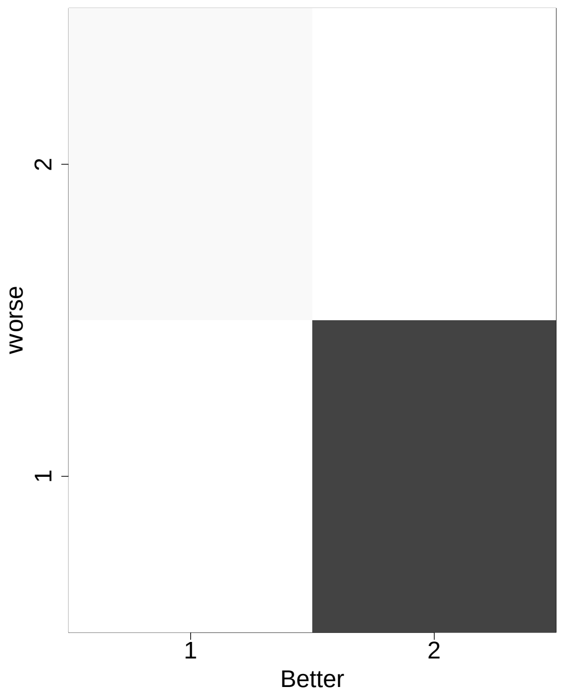
Metric	Statistical Test	EV	NE	NS
Epsilon	Fisher sign			
	Mann-Whitney			
Hypervolume	Fisher sign			
	Mann-Whitney			
R_2	Fisher sign			
	Mann-Whitney			

Figure 57: Interpreting results diagrams



Each grid cell is the p-value of a significance test. If the cell is completely black, then the a value will have to have been set < 0.05 in order to reject H0.
For the Fisher sign and Mann-Whitney tests the hypotheses are:

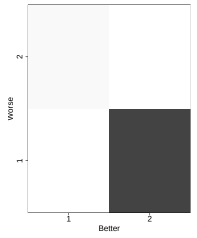
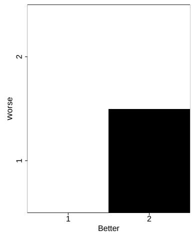
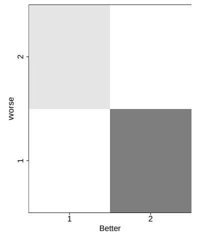
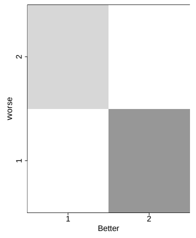
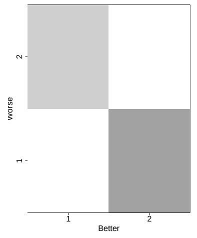
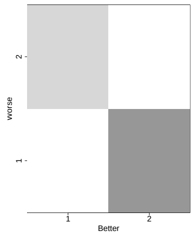
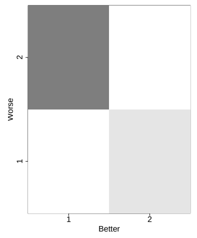
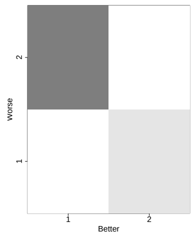
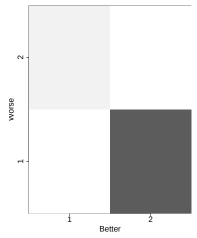
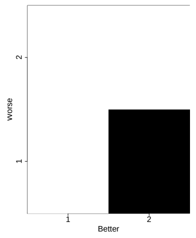
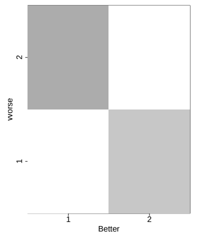
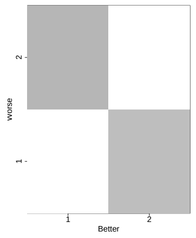
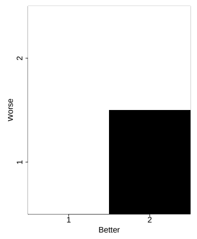
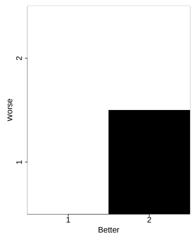
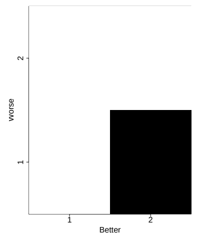
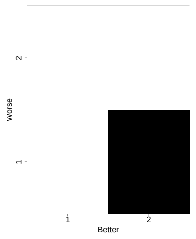
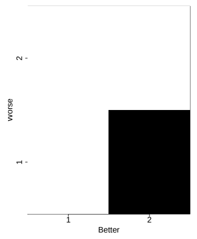
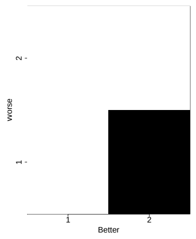
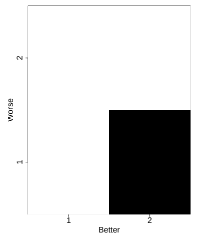
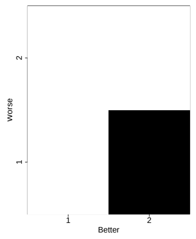
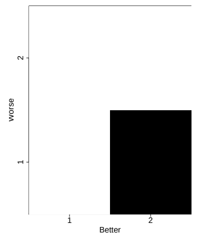
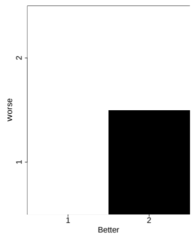
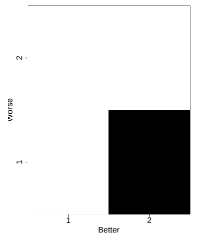
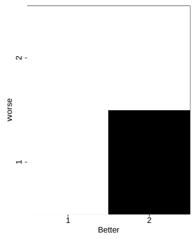
- H0: $M_{\text{better}} = M_{\text{worse}}$
- H1: $M_{\text{better}} < M_{\text{worse}}$

Where M is the population median of the Pareto metric calculated on 5 samples. A lower metric value is better than a higher metric value.
The correspondence between axis numbers and swarm types is:

1. SBS: SOMAS swarm chromosome is only evaluated on a single behavior and scenario.
2. MBS: SOMAS swarm chromosome is evaluated on multiple behaviors and scenarios.

Example: the above image shows that MBS outperforms SBS with statistically significance.

Table 27: Effectiveness of multi-scenario evolution for simple and complex behaviors

Behavior	Epsilon Metric		Hypervolume Metric		R_2 Metric	
	Fisher sign test	Mann-Whitney test	Fisher sign test	Mann-Whitney test	Fisher sign test	Mann-Whitney test
Intrusion Elimination						
DDoS Protection						
Competition						
Information Protection						

E.1 Object Oriented Design

The rationale behind object oriented design is to avoid redundancy in code and make code reuse as straightforward as possible. Consequently, there are three primary characteristics that define object oriented design.

- *Encapsulation:*

  All the necessary datastructures, variables, methods, and functions for a given item as grouped so that the user only needs to be aware of a shared interface and does not need to know any of the underlying details.

- *Loose coupling:*

  Coupling entails the degree that items depend on each other, and how much they know about their counterpart's internals. The items that are encapsulated are simplified as much as possible, so that they tend to have only one overall purpose, while keeping the dependencies between items low. For instance, an object oriented representation of a tooth brush has a single object for the bristles, instead of an object for each bristle. If there is an object for each bristle, then there is a large number of items that the handle has to be coupled with.

- *Polymorphism:*

  Polymorphism is defined by the principle of Liskov substitution. This principle states if one object can be substituted in the place of another for a certain function, for instance circle.volume() makes just as much sense as square.volume(), then they belong to a parent class, i.e. shape.volume(). Polymorphism is the calling of the same function on different types.

E.2 Properties Design Pattern

The properties design pattern is based on object oriented programming. There are two main differences between the two techniques. The first is that while the non-abstract class inheritance graph for object oriented programming is a forest, the inheritance graph for the properties design pattern is a graph. The second is that class structures can only be defined at compile time in object oriented programming (without specialized low level coding), whereas class structures can be define during runtime with the properties design pattern.

E.2.1 General Description. The properties design pattern is composed of two basic objects: a Thing and a Machine. Thing consists of a table that maps property names to objects. There are two kinds of accessors: direct and indirect. The combination of the two allows multiple Things to share the same property Object. The Machine is a subclass of Thing, which simply adds an execute function.

The properties design pattern is used as the fundamental concept of the systems SOMAS design, due to the two features differentiating it from object oriented design. Thus, it is a more general design and also allows certain methods described below that OO does not allow, or does not provide a simple and easy mechanism.

The properties model is also slightly augmented in that the execution of either a Thing or Machine method returns the Thing or Machine object. This allows operations to be chained together, with the possible inclusion of a glue procedure. This is an incorporation of Haskell's monad design pattern. The most straightforward benefit is conciseness of coding using the properties design pattern. A more theoretical benefit is that a sequence of operations can be treated as an entity in itself. Chaining together operations and using the chain as a first class object, i.e. can be passed and returned from functions, is a very effective functional programming technique.

E.2.2 Advantages.

- *Loose coupling:*

  Since all operations share the same basic class, it is simple to tie anything designed with the properties model into an external library. The interface with external libraries goes two ways: when used by the SOMAS system, a library operation is encapsulated within the Machine class; and when a SOMAS operation is used by an external library, it only needs to set the properties it is concerned with, which should encompass all similar potential libraries.

- *Bottom up programming:*

  When creating a certain operation, often it is easier to specify what is required than how it is acquired. The use of properties allows requirements to be a primary concern over acquirement. Properties are used in the operation as needed, and the properties can easily be artificially filled for testing purposes. This allows easy segmentation of code for testing purposes.

- *Runtime class specification:*

  A significant impediment of the traditional approach to OO as in Java and C++ is that it is usually impossible, or very difficult, to produce new classes during runtime. However, this is easy to do when a class is merely defined by a set of properties, since the properties can be defined at runtime.

- *Model checking (top down programming):*

  Since all objects are of the same type and all allow their properties to be accessed in the same way, it is possible to specify a general model checking framework. For instance, to check that a system has been implemented correctly, the sharing of properties between objects is validated by checking whether objects share the same object for a specific property. The existence of properties is simply validated by testing whether a given property returns a null. These basic techniques are amenable to further sophistication, such as testing whether a certain invariant holds before and/or after an operation. This is possible because all operations are also of the same basic type, and consequently can all be tested in the same way.

  The benefit of this model checking methodology is that it makes debugging simpler. It is also possible to use computational logic to deduce general properties of the system.

Furthermore, since, as mentioned, the design specifies a graph, graph theory can be used for model checking. For example, in some cases a partially ordered set is necessary. The problem with specifying the dependency relation in a partially ordered set is that a cyclical dependency can be created. Cycle detection can find such problems. General software design techniques can also be measure. For instance, if a system has very high connectivity, then it is not very loosely coupled and is likely to be badly designed.

E.2.3 Disadvantages. The properties are similar to Java's OO system, so there is redundancy in the implementation. Such redundancy also implies slower runtimes and greater memory usage.

Java and C++'s OO system is combined with many common integrated development environments, which can alert the programmer about errors in their code as they are writing it. Since the properties design pattern is not integrated with these same IDEs, their convenience can not be exploited to the same degree.

The code may be more verbose in some cases than if implemented directly using a traditional OO system.

- level 0 - single agent
  - change location
  - look for information
  - create information

    The mechanism of information transfer is part of the system that the agent is in, so this behavior encompasses both static and dynamic forms of information. I.e. pheromones and packet based communication.
  - change information
  - react to information
- level 1 - single agent
  - set waypoint

    Probably can consist of just setting a host IP, MAC, etc address
  - return to waypoint

    Use recorded host info to get routed back
  - release pheromone/message
  - look for pheromone/message
  - create decoy
  - create new agent
  - destroy agent
  - attack target
  - adapt to information

    This can be like a chameleon, so the agent looks like normal data. It can also be used to represent the kind of data it is most suited for, which works into the Characteristic Adaptation behavior.

    - specific signals
        * behavior trigger
        * behavior modification

          This is a meta operator where agents can directly modify each others rule set

- level 2 - multi agent There are only a small set of behaviors at this level, it is similar to level 0 in that it defines a basic behavior set.
    - triangulate (needs multiple nodes) This behavior is integral to many other mult-agent behaviors, since it provides a spacial overlay that can be used to calculate such behaviors as convergence or divergence.
        * position of other agents
        * position of targets
    - converge behavior

      Positive feedback, agents copying each other's behavior parameters. This also encompasses location and belief convergence.
    - diverge behavior

      Negative feedback, agents varying their behavior parameters based on agents' observations of each other This also encompasses location and belief divergence.
    - cluster

      This behavior can be described by something like k-means clustering or the like. This can be based on a system where agents are also pheromones, where their pheromone resemble some information signature (i.e. a specific file, or IP subnet), and they are attracted or repelled based on similarity. Also, this can be a combination of sync and diverge behaviors. In this case, the clustering takes place at a non physical level. Consequently, clustering can be considered a generic behavior that takes place at a number of different levels.

- level 3 - SO
    - Attack
        - Deceive
        - Lure (honey pot)
        - Ambush
        - Moving target discovery
    - Defense
        - Quarantine
        - Attack discovery
        - Identify abnormal behavior
    - Surveillance
        - Map network control points
    - Generic
        - Special ops behaviors

          For instance: formations, decoys, etc.
        - Discover rogue agent
        - Retreat from discovery
        - Characteristic adaptation

          A metaphor for this behavior is the bodywide signalling system, i.e. the nervous system. However, this is different than just plain communication. This is a high level behavior where agents are created that proliferate in a certain network. The interaction is bi-directional, so agents both adapt to information and change information.

      This general behavior can do a couple things:
        - Artificial immune system
        - QoS (certain agent adaptations retard the traffic, and others accelerate it) - using the wasp competition ranking

    * Create new swarm for new objective

- Distribute over network

G.1 Overview

The QuERIES methodology [30] uses three primary techniques:

- decision theory
- attack modelling with a POMDP model
- using markets to estimate probabilities for POMDPs

Assuming that the market effectiveness assumption holds such that a red team can accurately estimate the POMDP's transition and observation probabilities, the rest of the model appears to be a formally valid method for accomplishing risk analysis. The different costs in the decision grid are generally estimable, and a POMDP model, while it is PSPACE-hard [88] to derive an optimal policy, can still be used to generate approximate policies with techniques such as random point value iteration [121] and principle component analysis. These approximation techniques have shown success in a real world problem of robot navigation [122].

G.2 Need for Modelling Multiple Attacks

However, the general technique assumes attacks operate mutually independently of any other attack. While other attacks may be proceeding at the same time, they cannot have any influence on each other, otherwise the assumption behind the POMDP model does not hold. Since the QuERIES technique is being developed for the DoD and other owners of high value IP it is important that this assumption is accurat.

However, it is unlikely that attacks mounted against such customers will be isolated and independent. Infact, the current reports of the many and ongoing attacks against DoD show anything but isolated, mutually independent attacks. Instead, there are constant attacks occurring across the whole range of attack vectors, both orchestrated and independent.

In order to model such attacks, decentralized POMDP (DEC-POMDP) models [22] for the orchestrated attacks and interactive POMDP (I-POMDP) models [40] for the multiple independent attacks are necessary for accurate modelling. These models are even more intractable. NEXP-complete for DEC-POMDPs [22]. Unsolvable due to infinite belief nesting in the case of I-POMDPs [40]. Again, with certain assumptions and by using approximation algorithms it is possible to generate policies from such models. However, current research has not scaled to significant problem sizes yet.

G.3 Need for Adaptive Attack Vectors

Another problem is that POMDP models do not adapt themselves. During the course of an attack, an attacker is bound to change his attack method, and thus his states and state transition probabilities, as soon as defense operators begin to counteract what he is doing. While computational models are incapable of representing the full range of what humans can do in such a scenario, it is possible to at least improve on POMDP models by augmenting the models with a model transition function. In this way, the models can adapt themselves [56].

An issue with such adapted POMDP models is that they can be potentially Turing complete, depending on the nature of the model transition function. If the models become Turing complete, then it is impossible to generate a policy from them in the general case due to the halting problem.

G.4 Addressing Needs With Top Down Policy Generation

That being said, there is a tractable technique to generate approximate policies from augmented DEC-POMDP and I-POMDP models. The standard techniques for generating policies focus on a bottom up approach: the final solution is assembled from partial solutions using some kind of heuristic search algorithm.

It is also possible to generate policies using a top down approach where the space of complete solution is searched. The way this works is to first generate an arbitrary policy (which is a complete solution), evaluate it, then assemble new policies based the evaluated policies. This process repeats until a halting condition is reached, such as convergence or number of iterations. There are many existing algorithms that use this procedure, such as simulated annealing, evolutionary algorithms, tabu search, and ant colonies.

This technique allows policies to be generated that can be followed by multi-agent systems and are Turing complete, in a tractable amount of time [56]. Thus, the policies address the two needs of modelling multiple attackers and adaptive attack vectors.

The result is that top down policy generation using solution space search is capable of providing more realistic attack models for QuERIES.

Bibliography

1. "Autonomous Robot", February 2009. URL `en.wikipedia.org/wiki/Autonomous_robot`.
2. "File Transfer Protocol", february 2009. URL `wikipedia.org/wiki/Ftp`.
3. "Finite State Machine", february 2009. URL `wikipedia.org/wiki/Finite_state_machine`.
4. "Internet Protocol", february 2009. URL `wikipedia.org/wiki/Internet_Protocol`.
5. "The Law of Large Numbers", February 2009. URL `en.wikipedia.org/wiki/Law_of_large_numbers`.
6. "NS2", february 2009. URL `www.isi.edu/nsnam/ns`.
7. "OPNet", february 2009. URL `www.opnet.com`.
8. "OSI model", february 2009. URL `wikipedia.org/wiki/Osi_stack`.
9. "P (complexity)", february 2009. URL `wikipedia.org/wiki/P_(complexity)`.
10. "Transmission Control Protocol", february 2009. URL `wikipedia.org/wiki/Transmission_Control_Protocol`.
11. "Turing Completeness", february 2009. URL `wikipedia.org/wiki/Turing_comptele`.
12. Akass, Clive. "Sweden and Turkey in hacking war". *The Test Bed*, 2007.
13. Arabiya, Al. "Al Arabiya hit by Sunni-Shiite hacking war". *Al Arabiya*, 2008.
14. Arabiya, Al. "Sunni-Shiite hacking war disables 900 websites". *Al Arabiya*, 2008.
15. Bäck, Thomas. "Binary strings". *Evolutionary Computation 1 Basic Algorithms and Operators*. Institute of Physics Publishing, 2000.
16. Bäck, Thomas, D. B. Fogel, and T. Michalewicz (editors). *Evolutionary Computation 1 Basic Algorithms and Operators*. Springer, 2000.
17. Bak, Per, Chao Tang, and Kurr Wiesenfeld. "Self-organised criticality". *The American Physical Society*, 1988.
18. Baldassarre, Gianluca, Stefano Nol, and Domenico Parisi. "Evolving Mobile Robots Able to Display Collective Behaviors". *Artificial Life 9*. Massachusetts Institute of Technology, 2003.
19. Bar-Noy, Amotz, Samir Khuller, and Baruch Schieber. *The Complexity of Finding the Most Vital Arcs and Nodes*. Technical report, IBM Research Division and University of Maryland, 1995.

20. Barnes, Julian E. "Hacking could become weapon in US arsenal". *Los Angeles Times*, September 2008.

21. Berinato, Scott. "A Few Good Information Security Metrics". *CSO Security and Risk*, 2005.

22. Bernstein, Daniel S., Robert Givan, Neil Immerman, and Shlomo Zilberstein. "The Complexity of Decentralized Control of Markov Decision Processes". Need to find bibtex.

23. Bishop, Christopher M. *Pattern Recognition and Machine Learning*. Springer, 2006.

24. Bleuler, Stefan, Marco Laumanns, Lothar Thiele, and Eckart Zitzler. "PISA — A Platform and Programming Language Independent Interface for Search Algorithms". Carlos M. Fonseca, Peter J. Fleming, Eckart Zitzler, Kalyanmoy Deb, and Lothar Thiele (editors), *Evolutionary Multi-Criterion Optimization (EMO 2003)*, Lecture Notes in Computer Science, 494 – 508. Springer, Berlin, 2003.

25. Boschetti, F. and R. Gray. "Emergence and Computability". *Emergence: Complexity and Organization*, volume 9. The Complexity Society, the Institute for the Study of Coherence and Emergence, and Cognitive Edge, 2007.

26. Bradley, Tony. "Counter-Hacking : savior or vigilante". *About.com*, 2005.

27. Branke, J., Moez Mnif, Christian Miller-Schloer, Holger Prothmann, Urban Richter, Fabian Rochner, and Hartmut Schmeck. "Organic Computing : Addressing Complexity by Controlled Self-Organization".

28. Brooks, R. A. "Intelligence Without Representation". *Artificial Intelligence*, 139–159. 1991.

29. Camazine, S., J. Deneubourg, N. Franks, J. Sneyd, G. Theraulaz, and E. Bonabeau. *Self-Organization in Biological Systems*. Princeton University Press, 2003.

30. Carin, Lawrence, George Cybenko, and Jeff Hughes. "Cybersecurity Strategies: The QuERIES Methodology". *IEEE Computer*, 2008.

31. Carlin, Alan and Shlomo Zilberstein. "Value-Based Observation Compression for DEC-POMDPs". *Proceedings of the 7th International Conference on Autonomous Agents and Multi Agent Systems*. International Foundation for Autonomous Agents and Multi Agent Systems, 2008.

32. Cassandra, Anthony, Marian Rodine, Shilpa Bondale, Steve Ford, and David Wells. "Using POMDP-Based State Estimation to Enhance Agent System Survivability". *IEEE*, 2005.

33. de Castro, Leandro Nunes. *Fundamentals of Natural Computing (Chapman & Hall/Crc Computer and Information Sciences).* Chapman & Hall/CRC, 2006. ISBN 1584886439.

34. Çakar, Emre, Moez Mnif, Christian Müller-Schloer, Urban Richter, and Hartmut Schmeck. "Towards a Quantitative Notion of Self Organization". *IEEE*, 2007.

35. Center, SANS Internet Storm. "Survival Time", february 2009. URL `isc.sans.org/survivaltime.htmbl`.

36. Coello, Carlos A. Coello, Gary B. Lamont, and David A. Van Veldhuizen. *Evolutionary Algorithms for Solving Multi-Objective Problems*, chapter MOEA Parallelization. Springer, 2007.

37. Das, Subrata (editor). *Foundations of Decision-Making Agents: Logic, Probability, and Modality.* World Scientific Press, 2008.

38. Dembski, William. *Conservation of Information in Search.* Technical report, Center for Informatics, 2006.

39. Dewri, Rinku, nayot Poolsappasit, Indrajit Ray, and Darrell Whitley. "Optimal Security Hardening Using Multi-objective Optimization on Attack Tree Models of Networks". *CCS*, 2007.

40. Doshi, P. "On the Role Of Interactive Epistemology in Multiagent Planning". *Artificial Intelligence and Pattern Recognition.* 2007.

41. Doshi, Prashant and Piotr Gmytrasiewicz. *A Particle Filtering Algorithm for Interactive POMDPs.* Technical report, National Science Foundation, 2005.

42. Doziert, Gerry, Douglas Brownf, John Hurley, and Krystal Cainf. *Vulnerability Analysis of AIS-Based Intrusion Detection Systems via Genetic and Particle Swarm Red Teams.* Technical report, Auburn University and Clark-Atlanta University and The Boeing Company, 2004.

43. Eaton, John. "Octave", february 2009. URL `www.gnu.org/software/octave`.

44. Fisher, Dennis. "Storm, Nugache lead dangerous new botnet barrage". *Information Security*, 2007.

45. Foukia, Noria. "IDReAM: Intrusion Detection and Response Executed with Agent Mobility the Conceptual Model Based on Self- organizing Natural Systems". *ESOA*, 2004.

46. Frank, Eibe and Ian Witten. "Weka", december 2008. URL `www.cs.waikato.ac.nz/ ml/weka/index.html`.

47. Gagné, Christian and Marc Parizeau. "Genericity in Evolutionary Computation Software Tools: Principles and Case-Study". *International Journal on Artificial Intelligence Tools*, 15(2):173–194, April 2006.

48. Gmytrasiewicz, Piotr J. and Prashant Doshi. "Exact Solutions of Interactive POMDPs Using Behavioral Equivalence". *Association for the Advancement of Artificial Intelligence*, May 2006.

49. Group, Swarm Development. "SWARM", 1999. URL `www.swarm.org`.

50. Haken, H. *Information and Self-Organization: A Macroscopic Approach to Complex Systems*. Springer, 1988.

51. Hansen, EA, DS Berstein, and S Zilberstein. "Dynamic Programming for Partially Observable Stochastic Games". *Proceedings of the National Conference on Artificial Intelligence*. 2004.

52. Haykin, Simon. *Neural Networks: A Comprehensive Foundation*. Prentice Hall, 1999.

53. Heylighen, Francis. "The Science of Self-Organization and Adaptivity". *The Encyclopedia of Life Support Systems*. 2002.

54. Hinson, Gary. "Security metrics: more is not better". Internet, August 2008. URL `blog.isc2.org/isc2_blog/2008/08/security-metric.html`.

55. Hofstader, D. R. *Godel, Escher, Bach: An eternal golden braid*. The Harvester Press, Hassocks, Sussex, 1979.

56. Holloway, Eric. *Self Organized Multi Agent Swarms (SOMAS) for Accomplishing Network Goals*. Master's thesis, Air Force Institute of Technology, March 2009.

57. Hoschek, Wolfgang. "Colt", september 2004. URL `acs.lbl.gov/ hoschek/colt`.

58. Hughes, BD. *Random Walks and Random Environments*. Oxford University Press, 1995.

59. Igel, Christian and Marc Toussaint. "On Classes of Functions For Which No Free Lunch Results Hold". *Elsevier Science*, 2003.

60. Ihaka, Ross and Robert Gentleman. "R", december 2008. URL `www.r-project.org`.

61. Jennings, Nicholas R. and Michael Wooldridge. *Intelligent Agents: Theory and Practice*. Technical report, ESPRC, 1995.

62. Jennings, Nicholas R. and Michael Wooldridge. *Agent-Oriented Software Engineering*. Technical report, University of London, 2000.

63. Jesdanun, Anick. "US military prepares cyberwarfare offensive". *USA Today*, April 2008.

64. Kaelbling, L. P., M. L. Littman, and A. R. Cassandra. "Planning and acting in partially observable stochastic domains". *Artificial Intelligence*, 101:99–134, 1998.

65. Karrels, Daniel R. and Gilbert Peterson. "CyberCraft: Protecting Air Force Electronic Systems with Lightweight Agents". November 20007.

66. Kelso, J A Scott. "Phase Transitions and Critical Behavior in Human Bimanual Coordination". *American Physiological Society*, 1984.

67. Kerner, Sean Michael. "Google Gadget Under Attack at Black Hat". *Internet.com*, february 2009.

68. Kim, Jungwon and Peter J. Bentley. *Towards an Artificial Immune System for Network Intrusion Detection: An Investigation of Clonal Selection with a Negative Selection Operator*. Technical report, University College London, 2001.

69. Lamont, Gary. "A Comparison of NSGA-II and SPEA-2".

70. Lehn, Jean-Marie. *Supramolecular Chemistry - Scope and Perspectives - Molecules - Supermolecules - Molecular Devices*. Technical report, Institut Le Bel, Universit Louis Pasteur, 2987.

71. Li, M. and P. Vitanyi. *An Introduction to Kolmogorov Complexity and Its Applications*. Springer-Verlag, 1997.

72. Liu, Lei, Stefan Thanheiser, and Hartmut Schmeck. "A Reference Architecture for Self-organizing Service-Oriented Computing". *ARCS*, 205–219. Springer, 2008.

73. Liu, Yang and Kevin M. Passino. *Swarm Intelligence: Literature Overview*. Technical report, Ohio State University, 2006.

74. Lucas, J. R. "Minds, machines, and Gödel". *Philosophy*, 36, 1961.

75. Luke, Sean, Gabriel Catalin Balan, Keith Sullivan, and Liviu Panait. "MASON", june 2008. URL `cs.gmu.edu/ eclab/projects/mason`.

76. Luke, Sean, Liviu Panait, Gabriel Balan, Sean Paus, Zbigniew Skolicki, Elena Popovici, Keith Sullivan, Joseph Harrison, Jeff Bassett, Robert Hubley, and Alexander Chircop. "Java-based Evolutionary Computation Research System", May 2008. URL `www.cs.gmu.edu/ eclab/projects/ecj`.

77. Manna, Zohar. *Mathematical Theory of Computation*. McGraw Hill, 1974.

78. MathWorks. "MATLAB", february 2009. URL `www.mathworks.com`.

79. Matoušek, Petr, Jaroslav Ráb, Ondřej Ryšavý, and Miroslav Švéda. "A Formal Model for Network-wide Security Analysis". *15th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems*. 2008.

80. McMillan, Robert. "Black Hat: Web Browser Attack Skirts Corporate Firewall". *Computerworld*, 2007.

81. Metcalfe, R. “Metcalfe’s Law: A Network Becomes More Valuable As It Reaches More Users”. *Infoworld*, 1995.

82. Michalewicz, Zbigniew and David B. Fogel (editors). *How to Solve it: Modern Heuristics*. Springer, 1998.

83. Milton, J. Susan and Jesse C. Arnold. *Introduction to Probability and Statistics*, chapter Chapter 7 Estimation. McGraw Hill, 2003.

84. Mukkamala, Srinivas, Guadalupe Janoski, and Andrew Sung. “Intrusion Detection Using Neural Networks and Support Vector Machines”. *IEEE*, 2002.

85. Naveh, Barak. “JGraphT”, september 2008. URL `jgrapht.sourceforge.net`.

86. Newman, Mark. “The Physics of Networks”. *Physics Today*, November 2008.

87. Nolte, T., H. Hansson, and LL Bello. “Automotive Communications-Past, Current and Future”. *IEEE INternational Conference on Emerging Technologies*. 2005.

88. Norwig, P. and S. Russell. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2003.

89. Nowak, Dustin. *Exploitation of Self Organization in UAV Swarms for in Combat Environments*. Master’s thesis, Air Force Institute Of Technology, 2007.

90. Nowostawski, Mariusz, Martin Purvis, and Stephen Cranefield. “An Architecture for Self-organising Evolvable Virtual Machines”. *ESOA*. 2004.

91. Odlyzko, A. and B. Tilly. *A Refutation of Metcalfe’s Law and a Better Estimate for the Value of Networks and Network Interconnections*. Technical report, University of Minnesota, 2005.

92. O’Madadhain, Joshua, Danyel Fisher, Tom Nelson, Scott White, and Yan-Biao Boey. “JUNG”, July 2008. URL `jung.sourceforge.net`.

93. Packard, Katie. “Report Helps Define, Detect Insider Threats”. *AFCEA Signal Connections*, 2009.

94. Pamula, Joseph, Sushil Jajodia, Paul Ammann, and Vipin Swarup. “A Weakest-Adversary Security Metric for Network Configuration Security Analysis”. *ACM QoP*, 2006.

95. Panait, Liviu and Sean Luke. *Cooperative Mult-Agent Learning: The State of the Art*. Technical report, George Mason University, 2005.

96. Papazoglou, Mike P. and Willem-Jan van den Heuvel. *Service Oriented Architectures: Approaches, Technologies, and Research Issues*. Technical report, Tilburg University, 2005.

97. Payne, Shirley C. *A Guide to Security Metrics*. Technical report, SANS Institute, 2007.

98. Pearl. *Intelligent Search Strategies for Computer Problem Solving.* Addison Wesley, 1984.

99. Pohlheim, Hartmut. "GEATbx", July 2001. URL `www.geatbx.com`.

100. Price, Ian and Dustin Nowak. "Swarmfare", May 2005.

101. Prokopenko, M., F. Boschetti, and A. Ryan. "An Information-Theoretic Primer On Complexity, Self-Organisation And Emergence". *Advances in Complex System*, 2006.

102. Prokopenko, Mikhail, Vadim Gerasimov, and Ivan Tanev. *Measuring Spatiotemporal Coordination in a Modular Robotic System.* Technical report, CSIRO Information and Communication Technology Centre and Doshisha University and ATR Network Informatics Laboratories, 2005.

103. Rathnasabapathy, Bharaneedharan and Piotr Gmytrasiewicz. "Formalizing Multi-Agent POMDP's in the context of network routing". *Proceedings of the 36th Hawaii International Conference on System Sciences.* IEEE, 2003.

104. Reed, David P. "That Sneaky Exponential - Beyond Metcalfe's Law to the Power of Community Building". Context Magazine, 1999.

105. Reynolds, C. "Flocks, herds, and schools: A distributed behavioral model". *Computer Graphics*, volume 21, 25–34. 1987.

106. Rosenblatt, Joel. "Security Metrics: A Solution in Search of a Problem". *Educause Quarterly*, (3), 2008.

107. Rothlauf, Franz. *Representations for Genetic and Evolutionary ALgorithms.* Physica-Verlag, 2002.

108. Saltzer, J. H., D. P. Reed, and D. D. Clark. "End-To-End Arguments in System Design". *ACM Transactions on Computer Systems*, 1984.

109. Sanders, William H, Kaustubh R Joshi, Matti A Hiltunen, and Richard D Schlichting. *Infrastructure Reliability and Security Management Using Partially observable Markov Decision Processes.* Technical report, AT&T, 2006.

110. Schlegel, Tino, Peter Braun, and Ryszard Kowalczyk. "Towards autonomous mobile agents with emergent migration behaviour". *AAMAS '06: Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, 585–592. ACM, New York, NY, USA, 2006. ISBN 1-59593-303-4.

111. Schneier, Bruce. "The Nugache Worm/Botnet". *Schneier on Security*, 2007.

112. Searle, J. R. *The Rediscovery of the Mind.* MIT Press, 1992.

113. Seltzer, Larry. "The secret China-US hacking war". *eWeek*, March 2008.

114. Serugendo, Giovanna Di Marzo, Marie-Pierre Gleizes, and Anthony Karageorgos. "Self-Organization in Multi-Agent Systems". *The Knowledge Engineering Review*, 165–189. Cambridge University Press, 2005.

115. Serugendo, Giovanni Di Marzo, Marie-Pierre Gleizes, and Anthony Karageorgos. "Self Organization and Emergence in MAS: An Overview". *Informatica*. The Slovene Society Informatika, 2006.

116. Servat, David and Alexis Drogoul. "Combining amorphous computing and reactive agent-based systems: a paradigm for pervasive intelligence?" *AAMAS*, 441–448. ACM, 2002. URL `http://doi.acm.org/10.1145/544741.544842`.

117. Shalizi, Cosma Rohilla, Robert Haslinger, Jean-Baptiste Rouquier, Kristina Lisa Klinkner, and Cristopher Moore. "Automatic Filters for the Detection of Coherent Structure in Spatiotemporal Systems", July 29 2005. URL `http://arxiv.org/abs/nlin/0508001`.

118. Shannon, Claude E. "A mathematical theory of communication". *The Bell System Technical Journal*, 1948.

119. Siganos, Georgos, Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos. "Power Laws and the AS-Level Internet Topology". *IEEE/ACM Transactions on Networking*, 11(4), august 2003.

120. Smith-Sweeney, Brian. *Shifting Landscape of IT Security.* Technical report, NERCOMP, 2008.

121. Spaan, Matthijs T. J. and Nikos Vlassis. "Perseus: Randomized Point-based Value Iteration for POMDPs". *Journal of Artificial Intelligence Research*, 2005.

122. Spaan, MTJ and N. Spaan. "A Point-Based POMDP Algorithm for Robot Planning". *Proceedings of International Conference on Robotics and Automation.* IEEE, 2004.

123. Stein, William. "Sage", february 2009. URL `www.sagemath.org`.

124. Stokes, Jon. "Prearing for cyber warfare: US Air Force floats botnet plan". *Ars Technica*, May 2008.

125. Storn, Rainer, and Kenneth Price. "Differential Evolution - A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces". *Journal of Global Optimization*, 11:341–359, 1997.

126. Stytz, Martin R., Dale E. Lichtblau, and Sheila B. Banks. *Toward using intelligent agents to detect, assess, and counter cyberattacks in a network-centric environment.* Technical report, Institute for Defense Analysis, 2005.

127. Tanenbaum, Andrew S. and Maarten Van Steen. *Distributed Systems: Principles and Paradigms.* Prentice Hall, 2006.

128. Thomas, Tavaris J. *Fire Ant: An Algorithm Providing High-Performance Distributed Fault-Tolerant Communication for Internet-Scale Network Survivability.* Ph.D. thesis, Floriday State University, 2004.

129. Thomas, Timothy L. "Al Queda and the internet: the danger of cyberplanning". *Parameters*, 2003.

130. Tye, Michael. "Qualia", July 2007. URL `plato.stanford.edu/entries/qualia`.

131. Vollset, Einar. "Java Network Simulator", June 2007. URL `jns.sourceforge.net`.

132. Wall, Matthew. "GAlib", March 2007. URL `lancet.mit.edu/ga/dist`.

133. Weisstein, Eric W. "Fisher Sign Test". MathWorld-A Wolfram Web Resource, March 2009. URL `http://mathworld.wolfram.com/FisherSignTest.html`.

134. White, Tony and Bernard Pagurek. *Towards Multi-Swarm Problem Solving in Networks*. Technical report, Carleton University, 1999.

135. Williams, Thomas and Colin Kelley. "GnuPlot", march 2008. URL `www.gnuplot.info`.

136. Wilson, Tim. "DoS Gets Political In Estonia". *Dark Reading*, 2007.

137. Wolf, Tom De and Tom Holvoet. *Emergence and Self-Organisation: a statement of similarities and differences*. Technical report, Department of Computer Science, KULeuven, 2004.

138. Wolpert, David H. and William G. Macready. *No Free Lunch Theorems for Search*. Technical report, Santa Fe Institute, February 1995.

139. Wolpert, David H. and William G. Macready. *No Free Lunch Theorems for Optimization*. Technical report, Santa Fe Institute and TXN Inc., December 1996.

140. Yamins, Daniel. "The emergence of global properties from local interactions: static properties and one-dimensional patterns". Hideyuki Nakashima, Michael P. Wellman, Gerhard Weiss, and Peter Stone (editors), *AAMAS*, 1122–1124. ACM, 2006. ISBN 1-59593-303-4. URL `http://doi.acm.org/10.1145/1160633.1160837`.

141. Zhang, Zonghua and Pin-Han Ho. "Janus: A dual-purpose analytical model for understanding, characterizing and countermining multi-stage collusive attacks in enterprise networks". *Journal of Network and Computer Applications*, 32(3):710–720, 2009.

142. Zitzler, Eckart and Simon Künzli. "Indicator-Based Selection in Multiobjective Search". Xin Yao et al. (editors), *Parallel Problem Solving from Nature (PPSN VIII)*, 832–842. Springer-Verlag, Berlin, Germany, 2004.

Vita

Eric M. Holloway received a B.Sc. degree in computer science from Biola University, California, and graduated Summa Cum Laude. He is a Masters of Computer Science student at the Air Force Institute of Technology, as well as a communications officer in the United States Air Force.

Permanent address: 2950 Hobson Way
Air Force Institute of Technology
Wright-Patterson AFB, OH 45433

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704-0188*

1. REPORT DATE *(DD-MM-YYYY)*	19-03-09
2. REPORT TYPE	Master's Thesis
3. DATES COVERED *(From - To)*	August 2007 - March 2009
4. TITLE AND SUBTITLE	Self Organized Multi Agent Swarms (SOMAS) for Network Security Control
5a. CONTRACT NUMBER	
5b. GRANT NUMBER	
5c. PROGRAM ELEMENT NUMBER	
5d. PROJECT NUMBER	09-143
5e. TASK NUMBER	
5f. WORK UNIT NUMBER	
6. AUTHOR(S)	Holloway, Eric, M., First Lieutenant, USAF
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)	Air Force Institute of Technology Graduate School of Engineering and Management (AFIT/EN) 2950 Hobson Way, Building 640 WPAFB OH 45433-8865
8. PERFORMING ORGANIZATION REPORT NUMBER	AFIT/GCS/ENG/09-02
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	ATTN: National Security Agency Ms. Christine M. Nickell, Chief, Academic Outreach (I02E) I02E, Ste 6744, 9800 Savage Rd, Ft. Meade, MD 20755-6744 (comm) 410-854-6206, (fax) 410-854-7043 c.nicke2@radium.ncsc.mil
10. SPONSOR/MONITOR'S ACRONYM(S)	NSA/IASP
11. SPONSOR/MONITOR'S REPORT NUMBER(S)	
12. DISTRIBUTION/AVAILABILITY STATEMENT	APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.
13. SUPPLEMENTARY NOTES	

14. ABSTRACT

Computer network security is a very serious concern in many commercial, industrial, and military environments. This paper proposes a new computer network security approach defined by self organized agent swarms (SOMAS) which provides a novel computer network security management framework based upon desired overall system behaviors. The SOMAS structure evolves based upon the partially observable Markov decision process (POMDP) formal model and the more complex Interactive-POMDP and Decentralized-POMDP models, which are augmented with a new F(*-POMDP) model. Example swarm specific and network based behaviors are formalized and simulated. This paper illustrates through various statistical testing techniques, the significance of this proposed SOMAS architecture, and the effectiveness of self organization and entangled hierarchies.

15. SUBJECT TERMS

computer network security, cyberspace, cybercraft, agent swarms, self-organization, entangled hierarchies, multi-objective optimization, evolutionary computation

16. SECURITY CLASSIFICATION OF: a. REPORT	b. ABSTRACT	c. THIS PAGE	17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON	19b. TELEPHONE NUMBER *(Include area code)*
U	U	U	UU	240	Gary B. Lamont, PhD, AFIT/ENG	(937) 785-3636, x4718 (gary.lamont@afit.edu)

Reset